Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

The Media Sessions Markup Language (MSML) is used to control and invoke many different types of services on IP media servers. Clients can use it to define how media sessions interact on a media server and to apply services to individual or groups of users. MSML can be used, for example, to control media server advanced conferencing features, create sidebars, and insert media processing objects into media streams. As well, clients can use it with other languages such as the Media Objects Markup Language (MOML) or VoiceXML to interact with individual users or with groups of conference participants.

Table of Contents

1. Introduction.................................................................2
2. Overview...........................................................................3
   2.1 Objects.........................................................................4
1. Introduction

This document describes a markup language for manipulating media sessions within a media server. It is used to invoke many different types of services on individual connections, groups of connections, and conferences. It allows the creation of conferences, bridging different sessions together, and bridging sessions into conferences. It can be used to apply IVR operations and dialogs to sessions or conferences, and to modify the media flowing on a session. Dialogs may be specified in any suitable language from VoiceXML [7], which allows complete application interfaces to be executed by a media server, to Media Objects Markup Language (MOML) [9] which can be used to define individual user dialog commands and user input controlled widgets.
This work is influenced by the "Call Control Markup Language" (CCXML) [8] defined by the Voice Browser Working Group of the W3C. Where CCXML promotes the distribution of call control through scripting and conditional logic, MSML is focused as a transactional language. MSML also introduces more advanced conferencing features and application definable objects to modify media streams.

2. Overview

Media servers are a general purpose IP addressable platform for executing high density sophisticated media processing tasks. Applications are typically located separate from a media server and will often use SIP third party call control (3PCC) [2] to establish media sessions to the media server on behalf of end users. However MSML does not require that 3PCC be used.

Dynamic pools of media resources are available on media servers which allow application defined services to be created based upon the configuration and use of the 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 side bars. None of these change the SIP dialog or RTP session. Yet these do affect the media flow internal to the media server.

MSML is an XML language used to specify and change the flow of media streams within a media server. A network connection is established with the media server using SIP. Media received and transmitted on that connection will flow through different media resources on the media server dependent upon the requested service. Basic Network Media Services with SIP [6] defines conventions for associating a basic service with a SIP Request-URI. MSML allows services to be dynamically applied and changed by an application server during the lifetime of the SIP dialog.

MSML and MOML have been designed to work closely together: MOML addresses controlling and manipulating media processing operations (e.g., announcement, IVR, play and record, ASR/TTS, fax, video), while MSML addresses relationships of media streams (e.g., simple and advanced conferencing). Together, MSML and MOML create a general purpose media server control architecture. MSML can additionally be used to invoke other IVR languages such as VoiceXML.
MSML is currently defined for audio conferences. Other IETF work considers the more general case of multimedia conferences which consist of many different types of media processing resources. The requirements for media policy are described in [4]. Policy is manipulated using a Conference Policy Control Protocol (CPCP). The manipulation of media policy utilizes a graph which describes how media flows through the different media resources. Media policy manipulation in CPCP is described in [5].

2.1 Objects

A media object is an endpoint of a media stream. It may be something which terminates an external network connection or a resource which transforms or manipulates the media in some way. MSML defines several classes of media objects and allows other classes to be defined by applications using MOML or other suitable media resource description language. These are referred to as native and application classes and objects.

Native classes of media processing objects (with the class name in parentheses) are:

- network connection (conn);
- conference (conf);
- dialog execution context (dialog).

Network connection is an abstraction for the media processing resources involved in terminating a media stream from the network. They have a single full duplex media stream interface. Network connections get instantiated when a media session is established to a media server (e.g. through SIP).

Conference represents an n-1 audio mixer and the other media resources and state information required for an audio conference. They have multiple inputs and possibly several classes of inputs. Different classes of input allow the conference to offer different treatment to different audio streams. For example, an advanced conference may have some inputs contend to contribute to the mix while others are always able to contribute.

A conference has a single logical output consisting of the conference mix, less any contributed audio of an individual participant who receives the output. Conferences get instantiated using the <createconference> element. <createconference> and the other elements identified below are discussed in depth in the following section.

Dialog execution context is a generic reference to the set of resources which are used to create either a simple action, such as an atomic play or record operation, or for more complex application
interface components such as a VoiceXML interpreter. Dialog execution contexts are instantiated as a result of a <dialogstart> element. A dialog presents the same full duplex media stream interface as a network connection.

Application classes allow applications to define their own media processing objects and have them inserted into media streams where they are used to filter or transform the media stream. Application objects may be half duplex or full duplex. Half duplex objects reflect the common usage for things such as automatic gain control and filtering tones from conferences. However it may often be desirable to create full duplex objects which may treat the media differently in each direction.

Application defined objects get instantiated through the <insert> operation. MSML does not itself define the tools which describe these objects. Currently, MOML is supported for application class definition but other languages may also be used. MOML allows very simple or relatively sophisticated half and full duplex objects to be defined.

2.2 Identifiers

Objects are referenced using identifiers. When objects are first created, an MSML client may specify a name which will be used in the future to refer to that object. Future references to that object must use that name in a structured format as defined below. Objects which a client does not name will be assigned a name by a media server and returned to the client in the response to the command.

Object identifiers are a hierarchical list of semi-colon ';' separated terms. Terms may be either an object class or an object class followed by a colon ':' followed by qualifier which identifies an object within the class. The left most term of the identifier must identify a specific conference or connection in order to easily establish the scope of the identifier.

The identifier of a connection is the IP address and port number which a media server uses for the network media stream. Because a black hole address is used in some 3PCC scenarios, the network address in the received SDP can not be used. Applications may assign an alias to a connection or a set of connections. Aliases may be useful if an application needs to be able to refer to groups of connections which differ from the set of all connections joined to a conference.

There are two forms for a connection identifier; the network address form and the alias form. In the former case, the object class term of the identifier is the media server IP address and the qualifier is
the transport port number. In the latter case, the object class is "conn" and the qualifier is the name of the alias.

Dialogs contexts, and object instances of application defined classes, have identifiers relative to a connection or conference. For example, a VoiceXML application interface may be referenced as x.x.x.x;y;dialog to identify the dialog associated with network address x.x.x.x:y. If there were multiple dialogs associated with the same connection, then a qualifier would need to be specified along with the object class; e.g. x.x.x.x;y;dialog:foo. Similar to dialogs, application objects may be assigned qualifiers if more than one of the same class is to exist on the same media stream; e.g. for both input and output gain.

An identifier may be used to refer to multiple objects. For example, assume that an application has created a conference named "foo" and specified that each connection to the conference should use an application defined object class "autoGain" to adjust their audio to the same level. In this case, the identifier "conf:foo;autoGain" identifies all of the gain control objects for connections joined to the conference. The "autoGain" object for a specific connection would be referenced as "x.x.x.x:y;autoGain".

Although identifiers share a common syntax, MSML elements generally restrict the class of objects which 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 dialogs. Similarly, identifiers which refer to multiple objects are also only valid where expressly allowed by individual elements.

2.3 Media Flow

MSML allows clients to affect the flow of media streams internal to the media server. It allows clients to explicitly create conferences and to dynamically join and unjoin streams from those conferences. Streams may be directly joined to create simple ad hoc multi-party calls.

When a join requires that the outgoing half of a network connection have more than one input, the media server automatically bridges those inputs to create the media stream going out to the network. Not all traditional telephony platforms may support the ability to automatically bridge multiple inputs.

The maximum number of streams which may be automatically bridged is implementation specific but it is recommended that a media server support bridging at least two streams. Two reflects the ability to easily create simple three way calls and to bridge private announcements with a conference mix which can be heard by an
individual conference participant. Other applications with specialized topologies may benefit from more than two or three automatically bridged media streams. In the case of general conferences however, it is simpler to create a conference (with associated mixer) and then to join participants to the conference.

3. MSML Usage

SIP is used to create and modify media sessions with a media server according to the procedures defined in RFC 3261 [1]. Often, SIP third party call control will be used to create sessions to a media server on behalf of end users. MSML is used to define and change the service which a user connected to a media server will receive. As such, MSML clients are expected to be application servers which must have an authorized security relationship with the media server.

MSML transactions are originated based upon application domain events. 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.

Many alternatives exist for a transport mechanism for MSML. There may be one or many transport channels used to carry MSML based upon the requirements and structure of applications. SIP INVITE and INFO [3] requests and responses have been chosen to carry MSML in this initial release of the specification. INFO requests allow asynchronous mid-call messages within SIP with little additional semantics. There are existing widely deployed implementations of that method, it aids in initial developments which are closely coupled with SIP session establishment, and allows MSML to be directly associated with user dialogs when third party call control is used.

It is recognized that INFO is generally considered to not be a suitable general purpose transport mechanism for messages within SIP, although there have been proposals to make it more acceptable [10]. MSML is expected to evolve to more appropriate SIP usage and/or work with other protocols or as a stand alone protocol established through SIP, in future releases of this document.

Clients may wish to create SIP dialogs with the MS expressly for MSML. These dialogs would not have any associated media session. No media session is indicated by including SDP that has no media description (m= line) in the initial INVITE which establishes the dialog). One or more these dialogs may be created depending upon
application needs. Although initially INFO messages will be sent on this dialog, it is expected that this approach will be used to establish a separate MSML session in a future release.

4. Execution Flow

MSML assumes a model where there is a single control context within a media server. That context may have one or many SIP dialogs associated with it. It is assumed that any SIP dialogs associated with the control context of the media server have been authorized 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. 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 either be more natural for an application to request multiple actions at one time, or the application would like several actions to be closely coordinated by the media server. Multiple MSML elements received in a single request are 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 unjoining and joining one or more participants from a main conference to a sidebar conference. In the first scenario, network latencies may not be an issue but it is simpler for the application 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 insure that it has the necessary resources available to carry out the complete transaction before executing any elements of the request. Each element is expected to execute immediately. Elements such as <dialogstart> which take time are "forked" and executed in a separate thread. Once successfully forked, execution continues with the element following the dialog. As such, MSML does not provide mechanisms to sequence or coordinate other operations with dialog elements.

Processing within a transaction stops if any errors occur. Elements which were executed prior to the error are not rolled back. It is the responsibility of an application server 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 is returned in an error response. Note that errors which occur during the execution of a dialog occur outside the context of an MSML transaction.

Transaction results are returned as part of the SIP request response. The transaction results indicate the success or failure of the transaction. It will also include identifiers for any objects created by a media server which an application server did not name. Additionally, if the transaction failed, the reason for the failure is returned as well as an indication of how much of the transaction was executed before the failure occurred.

A media server using MSML also sends asynchronous events to an application server based on previous MSML requests. Events may be generated during the execution of a dialog created by a <dialogstart> element. An event is also generated when the execution of such a dialog terminates. The exact information which is returned is dependent upon the dialog language and the capabilities of the dialog execution environment. Events may also be generated to report requested information about the status of conferences.

5. Elements Received by a Media Server

5.1 <msml>

The root element for MSML. When received by a media server, it defines the set of actions which form a single MSML transaction. Actions are defined by the remaining elements of this section, most of which may appear zero or more times as children of <msml>.

Because MSML transactions execute immediately, the results of the transaction are included as a body in the response to the request. This response will contain any identifiers which the media server assigned to newly created objects. All messages which a media server generates to an application server are correlated to those object identifiers.

Attributes:

  version: "1.0" Mandatory

5.2 <alias>

Alias is used to assign a name to a connection. The same name may be aliased to multiple connections and a connection may have multiple aliases. Names allow multiple connections to be referenced at the same time in several circumstances.
attributes:

name: the name of the alias. Mandatory.

id: an identifier of a specific connection. Mandatory.

For example, if a conference has two moderators which will often hear the same set of private messages, an MSML client may create the alias "moderator" to refer to them as shown below.

```xml
<msml version="1.0">
  <alias name="moderator" id="192.0.2.4:6428"/>
  <alias name="moderator" id="192.0.2.6:9684"/>
</msml>
```

5.3 <join>

Join is used to create a media stream between two media objects. A media object may be a network connection or an internal media resource such as a conference. By default a full-duplex connection is created. However a half-duplex connection may be created by setting a duplex attribute to false. Join establishes the specified relationship between the two objects and does not change any pre-existing relationships.

Some media resources may only be half-duplex. It is illegal to join a full-duplex resource to a half duplex resource without specifying the join to be half-duplex. Otherwise the media server cannot be sure of the intent. At most one media stream may be created between the same two objects.

attributes:

id1: an identifier of either a specific connection or conference.

id2: an identifier of either a specific connection or conference.

duplex: "half" or "full". When "half" is specified the object identified by id1 receives media from the object identified by id2 but not vice versa. Default is full.

class: only allowed when either id1 or id2 identifies an advanced audio conference. These conferences have two types of audio inputs; "standard" participants contend to be one of the N-loudest which contribute to the conference mix; "preferred" always contribute to the conference mix. The duplex attribute
governs whether a media stream is actually established. Default is "standard". When both id1 and id2 are conferences, class is ignored and a preferred input is used.

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 media streams
supervisor: 192.0.2.2:4680
agent: 192.0.2.4:6428
customer: 192.0.2.6:9684

If all connections were already established on a media server the following would create the media flows previously described:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <join id1="192.0.2.2:4680" id2="192.0.2.4:6428"/>
  <join id1="192.0.2.4:6428" id2="192.0.2.6:9684"/>
  <join id1="192.0.2.2:4680" id2="192.0.2.6:9684" duplex="half"/>
</msml>
```

5.4 `<unjoin>`

Removes a media stream between two objects. `<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 specific connection or a conference. Mandatory.

id2: an identifier of either a specific connection or a conference. Mandatory.

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
</msml>
```
<unjoin id1="192.0.2.2:8" id2="conf:1"/>
<dialogstart target="conf:1" type="application/xml+moml">
  <play>
    <audio uri="file://leave_tone.wav"/>
  </play>
</dialogstart>
</msml>

5.5 <createconference>

This element is used to allocate media processing resources for audio conferences. Two types of audio conferences are defined; audio.basic and audio.advanced. A basic conference is simply an N-1 mix of the participants. An advanced conference supports the following features:

- N-loudest
- preferred speakers
- participant and conference resource reservations
- active speaker notifications
- sidebars

Characteristics of conferences may be changed dynamically during the conference by sending events to the conference identifier. The event name would identify the attribute to set and the namelist for the event would specify the new value. The type of conference cannot be changed once the conference has been created.

attributes: (all attributes are optional. Only the "id" attribute is allowed if type=audio.basic)

- type: audio.basic or audio.advanced. In the future this may be a URL which identifies a description of a conference. MOML may be a candidate language to specify conference descriptions. Default is audio.advanced.

- id: the name of the conference. If the attribute is not present, the media server will assign a globally unique name for the conference. Any events which the conference generates to an application will be correlated with this name.

- n: the number of participants who contend to be included in the conference mix based upon their audio energy. Default is 3.

- asn: boolean which defines whether active speakers should be reported. Only the participants who are active and eligible to contribute to the mix are reported. Default is false.
ri: the minimum reporting interval defines the minimum duration of time which must pass before changes to active speakers will be reported. Default is 5 seconds.

An example of creating a conference is shown below. This conference allows at most 2 participants to contend to be heard and reports the set of active speakers no more frequently than ten seconds.

<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <createconference id="myconf" n="2" asn="true" ri="PT10S"/>
</msml>

If the MSML client later decides that 3 participant should content to be mixed it may do so by:

<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <send target="conf:myconf" event="n" namelist="3"/>
</msml>

5.5.1 <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 which a media server will reserve for the conference. Any requests for resources beyond those which have been reserved should be honored on a best effort basis by a media server.

attributes:

  required: boolean which 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 true.

Two classes of resources are associated with a conference. Individual resources consist of those resources which are reserved for each participant. Shared resources are reserved for the entire conference. Examples of shared resources may be announcement players or voice recorders. Each class of resource is defined using the child elements <individual> and <shared> respectively. Each of these accepts an attribute which indicates the number of resource instances which are to be reserved.

The contents of each element describes the resources which are to be reserved for each instance. Descriptions are implementation
dependent. Media servers which support MOML may use the elements from that language as the basis for resource descriptions.

In the general case, the resources reserved for each participant must accommodate all of the codecs which a media server supports. However in some environments, the types of codecs which are expected may be known. In these cases, a media server may support descriptions which constrain reservations to different media formats.

For example, the following creates a conference and reserves individual resources for 20 participants, and reserves 2 instances of the shared resources for use on a conference wide basis.

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

The number of individual and shared reservations may be changed during the life of a conference by sending "reserve-ind" and "reserve-shared" events to the conference identifier. The namelist indicates the number of instances which are now desired. The success of the request is indicated in the response to the event. Requests for increases which cannot be honored do not change the reservation which was previously established.

Individual reservations are consumed when participants are joined to the conference. Applications must either include MSML indicating a <join> to the conference with the initial SIP INVITE, or must use the Request-URI conventions defined in [6], to insure that the conference connection may make use of the reserved resources when the connection is first established. Otherwise, if a dialog is required to determine the conference or service, the connection contends for available media server resources.

5.6 <destroyconference>

Deletes a conference. Any media streams between the conference and participants are removed.

attributes:

- id: the identifier for a conference. Mandatory.
term: a boolean value: When true, the media server will send a
BYE request on all SIP dialogs still associated with the
conference. Setting term equal to false allows applications to
start dialogs on connections once the conference has completed.
Default true.

5.7 <dialogstart>

Dialog start is used to instantiate application dialogs on network
connections or conferences. A dialog may consist of a simple atomic
commands or it may be an entire sequence of interactions. A media
server must support MOML to allow command driven interactions and
should support VoiceXML to allow execution of complex user
interfaces. Other dialog languages may also be supported.

Dialogs are executed in a separate execution environment or thread
from MSML. A media stream is created between the dialog execution
environment and the connection or conference. The media stream
between the dialog and the network connection or conference may be
full duplex or half duplex. When a dialog is performed on conferences
of type audio.advanced, the media stream is connected to the
conference mixer as a preferred speaker.

The dialog script may be specified either inline or by a URL. The
script must not be inline if the src attribute is specified. The
originator of the dialog is notified when the dialog completes. Any
results returned by the dialog when it exits are returned as part of
the completion notification.

attributes:

- target: an identifier of a connection or a conference which
  will interact with the dialog. A connection alias may be used
  if the application wishes multiple connections to share in the
  same dialog. In this case, a media server may create multiple
  bridged media streams as required. The dialog will fail if any
  of the streams cannot be established. Mandatory.

- src: the URL of the dialog script. Must not be used if the
dialog is inline.

- type: a MIME type which identifies the type of language with
  which the dialog is expressed. application/xml+moml and
  application/xml+vxml are used to identify MOML and VoiceXML
  respectively.

- id: a name which can be used to refer to the dialog. The
  attribute must not be included if the target is an alias. If
  the attribute is not present, the media server will assign a
globally unique name to each dialog. Any results which a dialog
sends to an application server will be correlated to its name.
Although dialogs may often be uniquely identified by their
target and the dialog class, a media server should always
return an identifier if none is supplied to allow for the case
of subsequent simultaneous dialogs. Simultaneous dialogs allow
features such as background recording or recognition.

duplex: "full" means the media stream is full duplex, "to"
means the media stream is half duplex to the dialog execution
environment (it receives audio), and "from" means the media
stream is half duplex from the dialog execution environment (it
sends media). Default "full".

The following example starts a VoiceXML dialog on a connection.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <dialogstart target="192.0.2.6:9684"
    type="application/xml+vxml"
    src="http://server.example.com/scripts/foo.vxml"/>
</msml>
```

5.8 <dialogend>

End dialog is used to terminate a dialog created through
<dialogstart> before it completes of its own accord. The operation of
<dialogend> depends upon 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 MOML, a "terminate" event will be
sent to the MOML context.

In both cases, the executing dialog has the opportunity to gracefully
complete and return data to the application server. As for all data
sent to the application server, it will be sent in a separate message
and correlated with the dialog identifier.

attributes:

id: the identifier of a dialog. Mandatory.

For example, the following would terminate the dialog started by the
previous example if it had not already completed.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <dialogend target="192.0.2.6:9684"/>
</msml>
```
If multiple dialogs were simultaneously active, then the "id" must include a qualifier identifying the specific dialog.

5.9 <insert>

Insert allows media streams to be modified by placing a media processing object between two existing objects. Inserted objects may be full duplex or half duplex. The existing media stream is replaced by a new stream which includes the inserted object. When a half duplex object is inserted, only the media stream in one direction is affected. Inserted objects are used to transform, filter, or otherwise modify a media stream. Examples are gain control and tone clamping for conferences but <insert> may be applicable for other capabilities as well.

Full duplex objects may not be symmetric in how they process media. Media in one direction may be treated or used differently from media in the other direction. For example automatic gain control may be applied to media going to a conference mix but a participant may have the ability to control the gain of the conference mix which they hear. In this case, the media server must know the orientation of the inserted object relative to the objects it is inserted between. Attributes "lhs" and "rhs" are used are used to name the left hand stream and the right hand stream of the object.

Media objects may be defined using MOML or other suitable media resource description languages. A media server which supports <insert> and <remove> must support MOML and may support other languages. Those languages must support the ability to identify the left hand and right hand streams of full duplex objects. Network connections, conferences, and dialog execution environments are examples of native media objects. Media objects may be referenced by a URL or defined inline.

Insert differs from using explicit unjoin and join operations because the inserted object is considered to be part of the media stream between the two original objects. If one of those objects terminates, for example a dialog, then the inserted object is automatically removed as part of deleting the media stream. Multiple objects may be inserted and all become part of the media stream.

It may often be desired to define the same treatment for all of the media streams associated with a conference. When one of the objects identifies a conference, <insert> allows the other object to be specified as "all" to indicate that a copy of the new object should become part of the media stream for all participants of the conference. The keyword "all" affects all current and future streams joined to the conference. The class of media stream that is affected
may be specified for conferences which support multiple kinds of
participants.

attributes:

id1: an identifier of the output of the inserted media object
or "all" if id2 specifies a conference. Mandatory.

id2: an identifier the input of the inserted media object or
"all" if id1 specifies a conference. Mandatory.

src: the URL of the media class. Must not be used if the dialog
is inline.

type: a MIME type which identifies the type of language used to
describe the media object. application/xml+moml must be
supported.

duplex: "half" or "full". When "half" is specified the inserted
object affects the stream in the direction from id2 to id1 but
not vice versa. Default is half.

lhs: a name which identifies the left hand stream of a full
duplex object. Media will flow from id2 to the input of the
right hand stream, through the inserted objected, and from the
output of the left hand stream, to id1.

rhs: a name which identifies the right hand stream of a full
duplex object. Media will flow from id1 to the input of the
left hand stream, through the inserted objected, and from the
output of the right hand stream, to id1.

objid: a name which can be used to refer to the object.

class: defines the class of conference media stream which is
affected when a conference supports multiple classes and the
"all" keyword is used. If no class is specified, then all media
streams are affected. The class attribute must not be used if
neither id1 or id2 specify "all" media streams.

For example, the following creates a conference, and specifies that a
gain control object from MOML should be inserted for each participant
that joins the conference that will automatically adjust the gain on
their input to the conference mix.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <createconference id="myconf"/>
  <insert id1="conf:myconf" id2="all"
```
5.10 <remove>

The remove element is used to remove objects which have been placed in a media stream using <insert>. Remove restores the original media stream.

attributes:

objid: the identifier of the object to remove. If objid refers to an object which was inserted into multiple media streams, then the object is removed from all applicable media streams. Mandatory.

5.11 <send>

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

attributes:

event: the name of an event. When the event is for an object defined using MOML (e.g. a dialog or an instance of an application defined class), then the name of the event should be prefixed with the MOML recipient. If the dialog or object instance consists of only a single MOML primitive, then the MOML recipient may be omitted. Mandatory.

target: an object identifier. Not all object classes or instances support events. Connections and basic conferences do not support events. VoiceXML supports events in a limited fashion. Advanced conferences support the "set" event to change their attributes. MOML dialogs and application objects defined using MOML support events based on their component MOML elements. Mandatory.

namelist: a list of zero or more parameters which are included with the event.
6. Elements Sent by a Media Server

6.1 <msml>

The root element for MSML. When sent by a media server, it encompasses the results of a request or the contents of events.

Attributes:

version: Mandatory "1.0"

6.2 <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 which initiated the transaction. The contents of the element may include text which expands on the meaning of the response.

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.

Two child elements allow for the response to include identifiers for objects created by the request which an application server did not name. Those elements are <confid> and <dialogid> for objects created though a <createconference> or <dialogstart> respectively.

6.3 <event>

The <event> element is used to convey an event to an application server. Two types of events are defined by MSML; dialog.exit, and conf.asn. These correspond to the termination of an executing dialog, and the notification of the current set of active speakers for a conference. Events may also be generated by an executing dialog. In this case the event type is specified by the dialog.

Events get sent to the URI from the Contact header of the SIP request which initiated the dialog execution or created the conference.

attributes:

name: the type of event. Mandatory.
id: the identifier of the conference or dialog which 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 which is associated with
the event.

7. Examples

7.1 MSML transaction

The following example creates a sidebar conference and moves three
participants from a main conference to the sidebar using <unjoin> and
<join>. It then does a half duplex join of the main conference to the
side bar so that those in the side bar may hear the main conference.
However a gain is inserted so that the volume of the main conference
does not intrude on the sidebar.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
  <createconference type="audio.basic" id="conf:sidebar"/>
  <unjoin id1="192.0.2.2:4680" id2="conf:main"/>
  <unjoin id1="192.0.2.4:6428" id2="conf:main"/>
  <unjoin id1="192.0.2.6:9684" id2="conf:main"/>
  <join id1="192.0.2.2:4680" id2="conf:sidebar"/>
  <join id1="192.0.2.4:6428" id2="conf:sidebar"/>
  <join id1="192.0.2.6:9684" id2="conf:sidebar"/>
  <insert id1="conf:sidebar" id2="conf:main" duplex="half" type="application/xml+moml">
    <gain amt="-30"/>
  </insert>
</msml>
```

7.2 Call Flow

The following call flow shows an application server (AS) using a
media server (MS) to create a three person conference. Three
terminals (T1, T2, T3) each dial in to a conferencing access number
hosted on the AS. The AS uses 3PCC to connect their media streams to
the MS and uses the MS to engage in a welcome dialog with each
person.

That dialog would consist of playing a greeting and asking them for a
conference identifier. Either MOML or VoiceXML could be used for
dialogs. This flow assumes that MOML has been used and that the
results are returned to the AS using MSML. Had VoiceXML been used,
the results would have been posted to the AS using HTTP.
After the first person has requested the conference, the AS instructs
the MS to create the conference and joins in that participant.
Subsequent participants are simply joined to the existing conference.
Each person records their name and the second and third participants
have their name announced to the conference as they join.
8. XML Schema

This schema needs to include that of MOML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="msml">
    <xs:complexType>
      <xs:choice>
        <xs:simpleContent>
          <xs:attribute name="name" type="xs:string"
            use="required"/>
          <xs:attribute name="id" type="xs:string"
            use="required"/>
        </xs:simpleContent>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="createconference" minOccurs="0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="reserve" minOccurs="0">
          <xs:complexType>
            <xs:choice maxOccurs="unbounded">
              <xs:element name="individual">
                <xs:complexType>
                  <xs:attribute name="num" type="xs:nonNegativeInteger"
                    use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
</xs:complexType>
</xs:element>
<xs:element name="shared">
 <xs:complexType>
  <xs:attribute name="num" type="xs:nonNegativeInteger" use="required"/>
 </xs:complexType>
</xs:element>
</xs:choice>
<xs:attribute name="required" type="xs:boolean" use="optional" default="true"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="type" type="conferenceType" use="optional" default="audio.advanced"/>
<xs:attribute name="id" type="xs:string" use="optional"/>
<xs:attribute name="n" use="optional" default="3">
 <xs:simpleType>
  <xs:restriction base="xs:nonNegativeInteger">
   <xs:minInclusive value="1"/>
   <xs:maxInclusive value="16"/>
  </xs:restriction>
 </xs:simpleType>
</xs:attribute>
<xs:attribute name="asn" type="xs:boolean" use="optional" default="false"/>
<xs:attribute name="ri" type="xs:duration" use="optional" default="PT5S"/>
</xs:complexType>
</xs:element>
<xs:choice maxOccurs="unbounded">
 <xs:element name="join">
  <xs:complexType>
   <xs:attribute name="id1" type="xs:string" use="required"/>
   <xs:attribute name="id2" type="xs:string" use="required"/>
   <xs:attribute name="duplex" type="duplexType1" use="optional" default="full"/>
   <xs:attribute name="class" type="classType" use="optional" default="audio"/>
</xs:complexType>
</xs:element>
</xs:choice>

<xs:element name="unjoin">
  <xs:complexType>
    <xs:attribute name="id1" type="xs:string"
                  use="required"/>
    <xs:attribute name="id2" type="xs:string"
                  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="destroyconference">
  <xs:complexType>
    <xs:attribute name="id" type="xs:string"
                  use="required"/>
    <xs:attribute name="term" type="xs:boolean"
                  use="optional" default="true"/>
  </xs:complexType>
</xs:element>

<xs:element name="dialogstart">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="moml-inline">
        <xs:attribute name="target"
                      type="xs:string"
                      use="required"/>
        <xs:attribute name="src"
                      type="xs:anyURI"
                      use="optional"/>
        <xs:attribute name="type"
                      type="xs:string"
                      use="required"/>
        <xs:attribute name="id"
                      type="xs:string"
                      use="optional"/>
        <xs:attribute name="duplex"
                      type="duplexType2"
                      use="optional"
                      default="full"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="dialogend">
  <xs:complexType>
    <xs:attribute name="id" type="xs:string"
                  use="required"/>
  </xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
<xs:element name="insert">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="moml-inline">
                <xs:attribute name="id1" type="xs:string" use="required"/>
                <xs:attribute name="id2" type="xs:string" use="required"/>
                <xs:attribute name="src" type="xs:anyURI" use="optional"/>
                <xs:attribute name="type" type="xs:string" use="required"/>
                <xs:attribute name="duplex" type="duplexType1" use="optional" default="half"/>
                <xs:attribute name="objid" type="xs:string" use="required"/>
                <xs:attribute name="class" type="classType" use="optional" default="standard"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<xs:element name="remove">
    <xs:complexType>
        <xs:attribute name="objid" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="send">
    <xs:complexType>
        <xs:attribute name="target" type="xs:string" use="required"/>
        <xs:attribute name="event" type="xs:string" use="required"/>
        <xs:attribute name="namelist" type="xs:string" use="optional"/>
    </xs:complexType>
</xs:element>
</xs:element>
</xs:sequence>
</xs:choice>

<xs:element name="result" type="resultType"/>
<xs:element name="event">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="name" type="xs:string"/>
      <xs:element name="value" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="exit" minOccurs="0"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:choice>

<xs:attribute name="version" type="xs:string" use="required" fixed="1.0"/>
</xs:complexType>
</xs:element>
</xs:complexType name="moml-inline">
<xs:sequence>
  <xs:choice maxOccurs="unbounded">
    <xs:element ref="group"/>
    <xs:element ref="play"/>
    <xs:element ref="record"/>
    <xs:element ref="dtmf"/>
    <xs:element ref="speech"/>
    <xs:element ref="vad"/>
    <xs:element ref="gain"/>
    <xs:element ref="agc"/>
    <xs:element ref="clamp"/>
    <xs:element ref="record"/>
    <xs:element ref="send"/>
  </xs:choice>
  <xs:element ref="exit" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:complexType name="resultType" mixed="1">
<xs:sequence>
  <xs:element name="confid" type="xs:string" minOccurs="0"/>
  <xs:element name="dialogid" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="response" type="xs:integer" use="required"/>
<xs:attribute name="mark" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
<xs:simpleType name="duplexType1">
  <xs:restriction base="xs:string">
    <xs:enumeration value="full"/>
    <xs:enumeration value="half"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="duplexType2">
  <xs:restriction base="xs:string">
    <xs:enumeration value="full"/>
    <xs:enumeration value="to"/>
    <xs:enumeration value="from"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="classType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="preferred"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="conferenceType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="audio.advanced"/>
    <xs:enumeration value="audio.basic"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

Security Considerations

MSML can be used to affect the service received by any user connected to a media server. The transport used to carry MSML must use appropriate security measures to insure that only authorized entities are able to use MSML.

References


Acknowledgments

Adnan Saleem and Yong Xin, both of Convedia, have provided significant insights, ideas, and contributions to this work; and Gilles Compienne and Ben Smith, both of Ubiquity Software, provided important feedback on a pre-release draft. The authors also wish to thank the other Convedia partners and customers that supplied valuable input into and review of this specification.
Authors’ Addresses

Tim Melanchuk
Convedia
4190 Still Creek Drive, Suite 300
Vancouver, BC, V5C 6C6
Canada

email: timm@convedia.com

Garland Sharratt
Convedia
4190 Still Creek Drive, Suite 300
Vancouver, BC, V5C 6C6
Canada

email: gsharratt@convedia.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF’s procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Copyright (C) The Internet Society 2003. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.