Dialogic®

MSML Media Server Software

User's Guide

February 2008

05-2513-003

Publication Date: February 2008

Document Number: 05-2513-003

Contents

Figures

Tables

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2513-003	February 2008	High-Level Feature Support by Platform table: Updated to reflect PSTN support for Dialogic® HMP Software 3.0WIN and Dialogic® HMP Software 3.1LIN and Video support for Dialogic® HMP Software 3.0WIN.
05-2513-002	December 2007	About This Publication chapter: Updated Scope section (audio conferencing is supported). Configuration chapter: Updated media_server Application section and mediaserver Configuration Script section to identify Linux specific information and add Windows information. Updated Configuring Schema Validation section to include msml.xsd and information on enabling/disabling feature on Windows® systems. Sample Use Case chapter: Added I-frame information to Step 5G in Record Message section. Updated Audio/Video Play/Record Scenario figure to include step 5G. Feature Support chapter: Added Feature Highlights by Platform section. Renamed title of MSML and MOML sections from "...Feature Support Matrix" to "...Protocol Support Matrix". Feature Details chapter: New chapter. Includes SIP Session Timer, SIP re-INVITE, and HTTP Play and Record (the last section also includes HTTP requirements information which was moved from Feature Support chapter).
05-2513-001	October 2007	Made global changes to reflect Dialogic brand. Changed title from "MSML/MOML Media Server Software User's Guide" to "MSML Media Server Software User's Guide" and removed certain references to MOML within the document. Configuration chapter: Added new section: Configuring Schema Validation. MOML Feature Support Matrix table: Changed the postspeech attribute in the <record> section to supported.
05-2513-001-03	August 2006	General: Updates for "http://" scheme support. MSML Feature Support Matrix table: Updated to reflect conference support. Deviations from IETF Draft chapter: Added two items related to conference support. Configuration File Format section: Added MEDIA_CODER component for RTF logging. Client Categories section: Updates to reflect conference support. Conference Elements section: Added brief descriptions of the conference elements. Media Server Object Model section: Added a subsection describing the "Conference" object.
05-2513-001-02	June 2006	Updates for diagnostics integration with RTF.
05-2513-001-01	April 2006	Initial version of document.

About This Publication

The following topics provide information about this publication.

- Purpose
- Scope
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This publication documents the Media Server Markup Language (MSML) Media Control Interface software that provides an interface between an application server (AS) and a media server (MS).

This publication is for users of the MSML Media Server Software who write applications that require remote control of MS resources.

Scope

The MSML Media Server Software functionality is being introduced in a phased approach. This manual documents the functionality provided by the current implementation phase, which includes support for:

- MSML Core Module
- MSML Stream Management Module (audio conferencing is included)
- MSML Dialog Module
- MOML Core Module
- MOML Group Module (parallel topology only)
- MOML Basic Primitives Module
- MOML Transform Primitives Module (gain only)

Functionality that is not supported by the current implementation phase includes:

- MSML Stream Management Module (video conferencing)
- MOML Speech Module
- MOML Fax Module

Future implementation phases are planned to provide additional MSML support.

Intended Audience

This publication is for:

- System Integrators
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

This publication assumes that the reader is familiar with the Session Initiation Protocol (SIP).

How to Use This Publication

This manual is divided into the following sections:

- Chapter 1, “MSML Media Server Software Overview” describes the role of the MSML Media Server Software in a Media Server environment.
- Chapter 2, “Configuration” explains how to configure the MSML Media Server Software for operation on a Media Server.
- Chapter 3, “Sample Use Case” presents an application that demonstrates many of the features currently supported by the MSML Media Server Software.
- Chapter 4, “Feature Support” specifies high-level feature support by platform and identifies which elements (as documented in the MSML and MOML IETF drafts) are currently supported or not supported by the MSML Media Server Software.
- Chapter 5, “Feature Details” provides details on features supported, including a feature description and how-to information.
- Chapter 6, “Deviations from IETF Draft” explains deviations from the MSML and MOML IETF drafts.
- Chapter 7, “Diagnostics” describes the logging capabilities available to the MSML Media Server Software for diagnostic purposes.
- Appendix A, “Media Server Markup Language (MSML) Overview” provides a high-level introduction to MSML.
- Appendix B, “Media Object Markup Language (MOML) Overview” provides a high-level introduction to MOML.
- A **Glossary** can be found at the end of the document.

Related Information

See the following for additional information:

- *http://www.dialogic.com/manuals/* (for Dialogic® product documentation)
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

MSML Media Server Software Overview 1

This chapter provides an overview of the MSML Media Server Software. Topics include:

1.1 Introduction

The MSML Media Server Software is an integral part of the system software provided by Dialogic (for example, Dialogic® Host Media Processing (HMP) Software or Dialogic® Multimedia Software for AdvancedTCA).

When the Dialogic® system software is installed on a media server (MS), the MSML Media Server Software enables a remote client, also known as an application server (AS), to control media resources.

Note: The MSML Media Server Software is based on the evolving Media Server Markup Language (MSML) as defined in the MSML IETF Draft, saleem-msml-00, which combines the original MSML and Media Object Markup Language (MOML) drafts. Little to no change was introduced in this draft when compared to the separate MSML and MOML version 6 drafts.

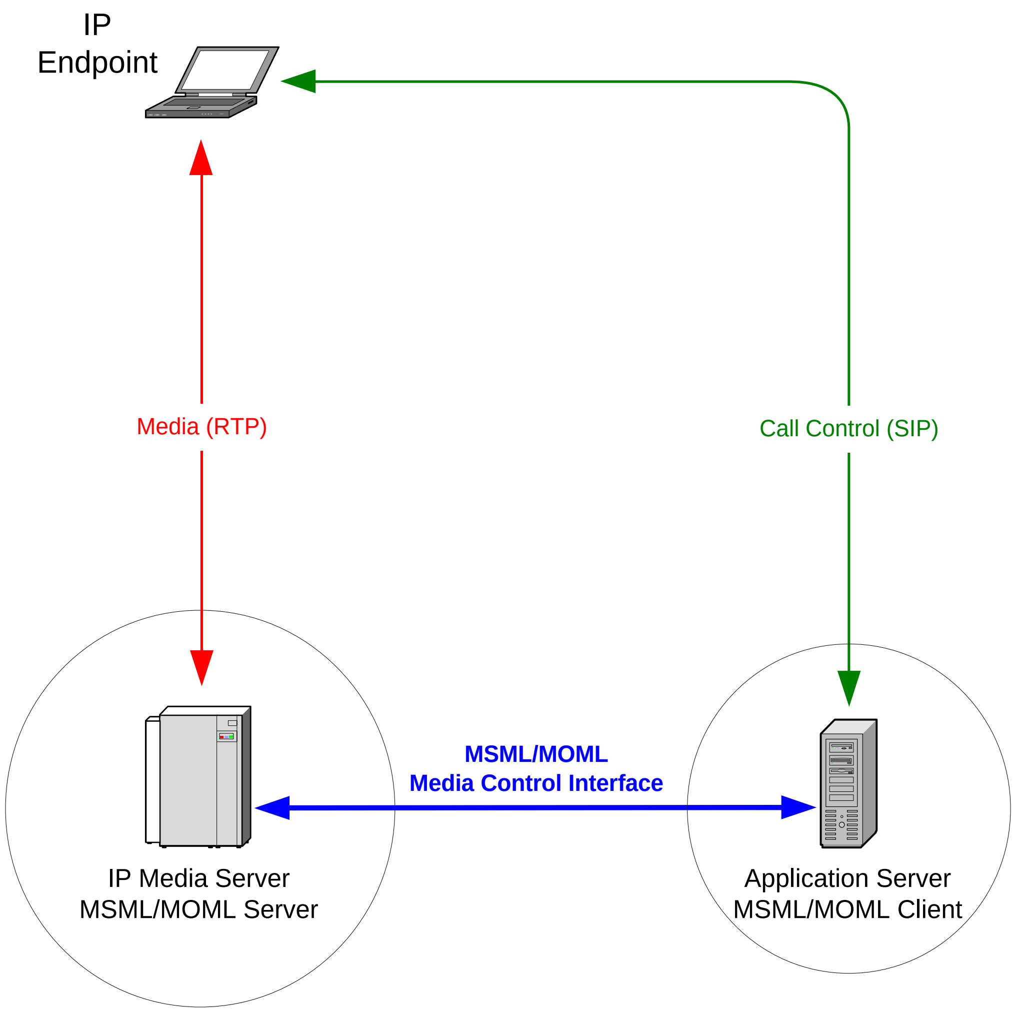
The connection between the AS and MS is established using the SIP protocol; thereafter, media control commands/responses (in the form of MSML control syntax) are exchanged in SIP messages, such as the INFO message or the 200 OK response.

1.2 Media Server Operating Model

Figure 1 shows an environment where the media server (MS) and application server (AS) operate as separate entities. The MSML Media Server Software runs on the MS and provides the interface between the AS and the MS as shown. The MS is responsible for media processing only; call control is the responsibility of the AS.

The AS, as a MSML client, must be capable of interpreting and generating MSML control syntax and must support the SIP INVITE, 200 OK, ACK, BYE and INFO messages.

Figure 1. Media Server Operating Environment


IP Endpoint
Media (RTP)
Call Control (SIP)
MSML/MOML Media Control Interface
IP Media Server MSML/MOML Server
Application Server MSML/MOML Client

Configuration 2

This chapter discusses configuration topics, such as how to configure a media server to use the MSML Media Server Software.

2.1 media_server Application

The *media_server* application runs on the media server and provides the MSML Media Server Software functionality.

On Linux, the *media_server* application can be configured using the *mediaserver* configuration script as described in Section 2.2, "mediaserver Configuration Script", on page 13.

On Windows®, the *media_server* application runs as a Windows® service.

2.2 mediaserver Configuration Script

On Linux, the *mediaserver* configuration script is used to enable/disable and start/stop the *media_server* application and is located in the *bin* directory.

The *mediaserver* configuration script provides the following options:

mediaserver enable
: Enables the *media_server* application. All subsequent calls to *dlstart* start the *media_server* application. All subsequent calls to *dlstop* stop the *media_server* application.

mediaserver disable
: Disables the *media_server* application. Subsequent calls to *dlstart* do **not** start the *media_server* application.

mediaserver start
: Starts the *media_server* application. The *media_server* application must be started after *dlstart* successfully completes.

mediaserver stop
: Stops the *media_server* application.

2.3 Configuring Schema Validation

When enabled, the schema validation functionality included in the product validates each and every XML body received from the application server against a pre-installed MSML schema, *msml.xsd*, located in the *cfg* directory. The inbound XML body must pass the schema validation before it will be executed by the media server. This is especially useful for application developers during the development process. It ensures that the syntax and attribute definitions in the XML body are correct and match the supported functionality in the schema.

To reduce CPU utilization, this functionality is disabled by default. It is recommended that it remain disabled in a production runtime environment.

To enable this functionality on Linux operating systems, specify the `conf msml schema-validation` command using the Command Line Interface (CLI) software.

To enable this functionality on Windows® operating systems, edit the *media_server.xml* configuration file, which is located in the *cfg* directory.

Sample Use Case 3

This chapter describes a simple application that demonstrates many of the capabilities provided by the current version of the MSML Media Server Software. Topics include:

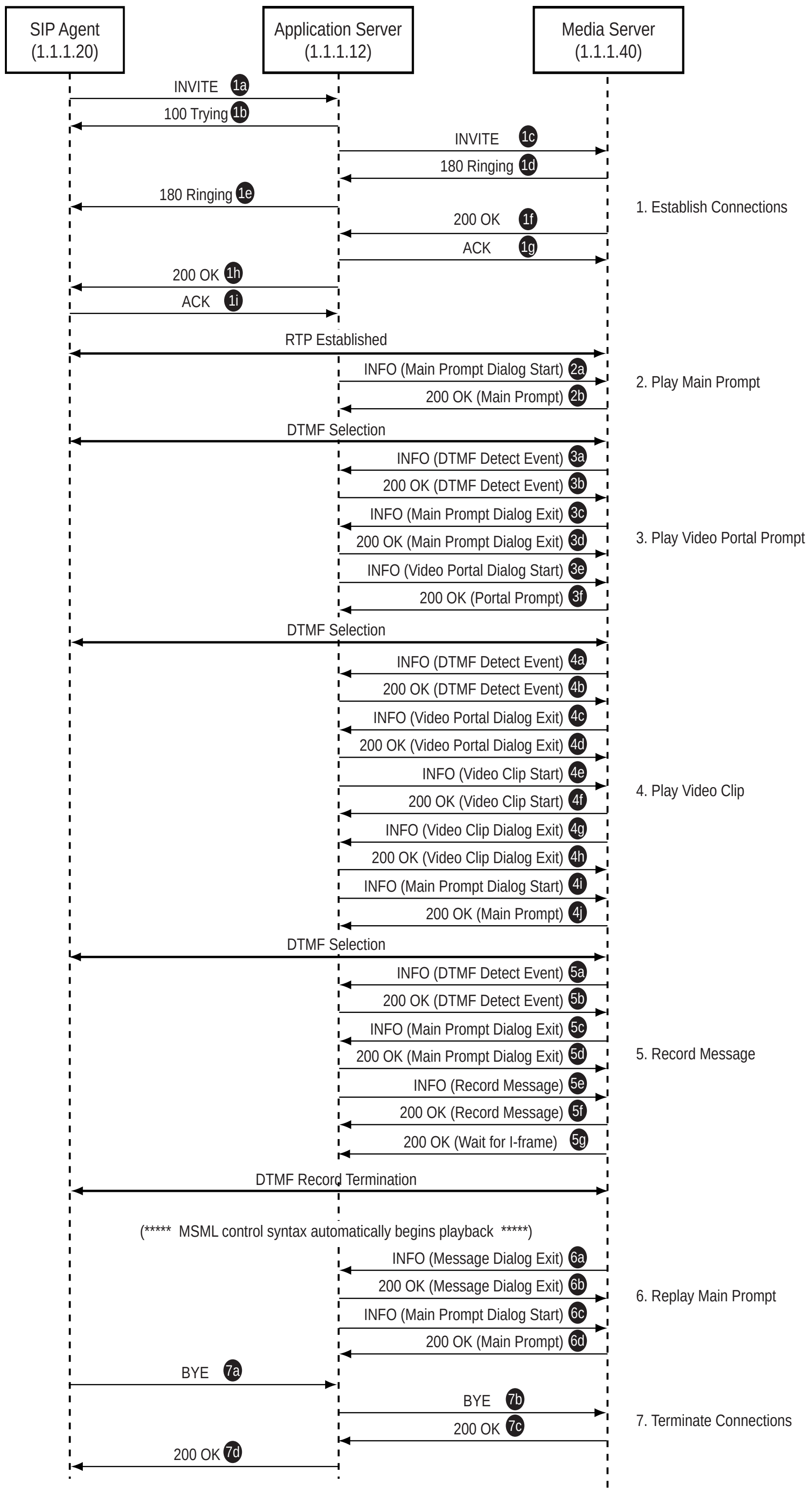
3.1 Use Case Description

In this application, the user is presented with options to play and record audio and video clips/messages. The application server (AS) communicates with the media server (MS) using the MSML Media Server Software to perform the selected operations. Figure 2 shows the exchange of SIP messages between the SIP client, AS, and MS to perform the functionality required. Many of the messages exchanged between the AS and MS include MSML control syntax that are interpreted and acted upon by the MSML Media Server Software.

The following sequence describes the high-level activities from the SIP client and MS perspectives:

1. The SIP client initiates a SIP dialog with the AS and a media session with the MS.
2. The MS plays the **Main Prompt** with options for the playing of prerecorded clips of News, Weather, Messages, Image of Your Daily Schedule, or the recording of an audio-visual message. The MS then waits on DTMF detection. The MS waits forever and never disconnects, unless a BYE is issued or unless AS timers set a limit on call length.
3. The SIP client makes a selection using DTMF. The selection is to display the **Video Portal Prompt**.
4. The MS plays the **Video Portal Prompt**.
5. The SIP client makes a selection using DTMF. The selection is to play a video.
6. The MS plays the selected video clip to completion.
7. The MS plays the **Main Prompt**.
8. The SIP client makes a selection using DTMF. The selection is to record, then play back, a video message.
9. The MS records the video message.
10. The SIP client stops the recording of the video message with any DTMF.
11. The MS starts the playback of the recorded video message (executed in MSML syntax).
12. The MS plays the recorded video message to completion.
13. The MS plays the **Main Prompt**.
14. The SIP client disconnects.
15. The MS disconnects.

Figure 2. Audio/Video Play/Record Scenario


SIP Agent (1.1.1.20)
Application Server (1.1.1.12)
Media Server (1.1.1.40)
INVITE 1a
100 Trying 1b
INVITE 1c
180 Ringing 1d
180 Ringing 1e
200 OK 1f
ACK 1g
200 OK 1h
ACK 1i
1. Establish Connections
RTP Established
INFO (Main Prompt Dialog Start) 2a
200 OK (Main Prompt) 2b
2. Play Main Prompt
DTMF Selection
INFO (DTMF Detect Event) 3a
200 OK (DTMF Detect Event) 3b
INFO (Main Prompt Dialog Exit) 3c
200 OK (Main Prompt Dialog Exit) 3d
INFO (Video Portal Dialog Start) 3e
200 OK (Portal Prompt) 3f
3. Play Video Portal Prompt
DTMF Selection
INFO (DTMF Detect Event) 4a
200 OK (DTMF Detect Event) 4b
INFO (Video Portal Dialog Exit) 4c
200 OK (Video Portal Dialog Exit) 4d
INFO (Video Clip Start) 4e
200 OK (Video Clip Start) 4f
INFO (Video Clip Dialog Exit) 4g
200 OK (Video Clip Dialog Exit) 4h
INFO (Main Prompt Dialog Start) 4i
200 OK (Main Prompt) 4j
4. Play Video Clip
DTMF Selection
INFO (DTMF Detect Event) 5a
200 OK (DTMF Detect Event) 5b
INFO (Main Prompt Dialog Exit) 5c
200 OK (Main Prompt Dialog Exit) 5d
INFO (Record Message) 5e
200 OK (Record Message) 5f
200 OK (Wait for I-frame) 5g
5. Record Message
DTMF Record Termination
(***** MSML control syntax automatically begins playback *****)
INFO (Message Dialog Exit) 6a
200 OK (Message Dialog Exit) 6b
INFO (Main Prompt Dialog Start) 6c
200 OK (Main Prompt) 6d
6. Replay Main Prompt
BYE 7a
BYE 7b
200 OK 7c
200 OK 7d
7. Terminate Connections

3.2 MSML Control Syntax in Use Case

Figure 2 includes labels to identify the SIP messages exchanged among the SIP client, AS, and MS. For easier reference, the main steps are designated with numbers and subordinate steps are designated with lowercase letters. The following sections describe the steps (and subordinate steps) with particular emphasis on the MSML control syntax included in the exchanged messages. The main steps are:

- Establish Connections
- Play Main Prompt
- Play Video Portal Prompt
- Play Video Clip
- Record Message
- Replay Main Prompt
- Terminate Connections

Note: In the subsections following, the first SIP INFO message (in Section 3.2.2) is shown in its entirety to highlight the fact that the AS must set the "Content-Type" and "Content-Length" in the SIP header. For the remaining SIP messages, the SIP headers are not included since the focus is on the MSML control syntax.

3.2.1 Establish Connections

Steps 1a to 1i

These steps comprise standard SIP message exchange for the establishment of SIP dialogs between the SIP client and AS and between AS and MS and the establishment of a media session (RTP) between the SIP client and MS. It is over the RTP connection that the user responds to prompts using DTMF selections. There is no MSML control syntax involved in this message exchange.

However, one important piece of information received by the AS in **Step 1f** is the **network connection identifier** that is assigned by the MS. The identifier is the "tag" value included in the "To" header of the SIP 200 OK response to the initial INVITE sent by the AS. In this example, the "To" header is:

```
To:<sip:1.1.1.12>;tag=b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2
```

In MSML control syntax, the connection identifier is specified as "conn:<tag value>". In the sample control syntax in Section 3.2.2 to Section 3.2.6, the network connection identifier is shown in **bold** text.

3.2.2 Play Main Prompt

Step 2a

The AS sends the MS an INFO message to play the **Main Prompt** dialog. The complete INFO message shown below includes MSML control syntax:

```
INFO sip:AS@1.1.1.40:5060 SIP/2.0
Call-ID: ae11d01e-7350-462d-8a9e-169c79d7361a@1.1.1.20
From: "Administrator" <sip:WINDOWS-E6UOEQY>;tag=-1179327240.1.babababamaggmjjhbgpgjfogkj
To: <sip:1.1.1.12>;tag=b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2
CSeq: 3 INFO
Contact: sip:1.1.1.12:5060
Content-Type: text/xml;charset=UTF-8
Content-Length:709
Max-Forwards: 70
Via: SIP/2.0/UDP 1.1.1.12:5070;branch=z9hG4bK0101010CBADF00D00000109F178B4485

<?xml version="1.0" encoding="UTF-8" ?>
<msml version="1.0">
<dialogstart target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2"
type="application/moml+xml">
   <group topology="parallel" >
      <play>
         <media>
            <video uri="file://./av/main_menu.vid" format="video/raw:codecs=h263" />
            <audio uri="file://./av/main_menu.pcm" format="audio/pcm:codecs=mulaw"
                        audiosamplesize="8" audiosamplerate="8"/>
         </media>
      </play>
      <collect iterate="forever" cleardb="true">
         <detect>
            <send target="source" event="done" namelist="dtmf.digits" />
            <send target="group" event="terminate"/>
         </detect>
      </collect>
   </group>
</dialogstart>
</msml>
```

Step 2b

The MS sends a 200 OK response to indicate success. The 200 OK response includes the following MSML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:10</dialogid>
</msml>
```

3.2.3 Play Video Portal Prompt

Step 3a

At this point, the SIP client makes a selection that is transmitted as DTMF to the MS. The selection is to display the **Video Portal Prompt** dialog. The MS sends the AS an INFO message that includes the following MSML control syntax describing the DTMF Detect event.

```
<msml version="1.0">
   <event name="done"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:10">
      <name>dtmf.digits</name><value>1</value>
   </event>
</msml>
```

Step 3b

The AS sends a 200 OK response to the MS to acknowledge successful receipt of the DTMF Detect event. The 200 OK response does not includes any MSML control syntax.

Step 3c

The MS sends the AS an INFO message with the following MSML control syntax to indicate that the **Main Prompt** dialog is exiting.

```
<msml version="1.0">
   <event name="msml.dialog.exit"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:10"/>
</msml>
```

Step 3d

The AS sends a 200 OK response to the MS to acknowledge **Main Prompt** dialog exit. The 200 OK response does not contain any MSML control syntax.

Step 3e

The AS sends the MS an INFO message to start playing the **Video Portal Prompt** dialog. The INFO message includes the following MSML control syntax:

```
<?xml version="1.0" encoding="UTF-8" ?>
<msml version="1.0">
<dialogstart target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2"
      type="application/moml+xml">
   <group topology="parallel" >
      <play>
         <media>
            <video uri="file://./av/vportal_menu.vid" format="video/raw:codecs=h263" />
            <audio uri="file://./av/vportal_menu.pcm" format="audio/pcm:codecs=mulaw"
                  audiosamplesize="8" audiosamplerate="8"/>
         </media>
      </play>
      <collect iterate="forever" cleardb="true">
         <detect>
            <send target="source" event="done" namelist="dtmf.digits dtmf.len dtmf.end
                  dtmf.last" />
            <send target="group" event="terminate"/>
         </detect>
      </collect>
   </group>
</dialogstart>
</msml>
```

Step 3f

The MS sends a 200 OK response to the AS to acknowledge the starting of the **Video Portal Prompt** dialog. The 200 OK response includes the following MSML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:13</dialogid>
</msml>
```

3.2.4 Play Video Clip

Step 4a

At this point, the SIP client makes a selection that is transmitted as DTMF to the MS. The selection is to play a **Video Clip**. The MS sends the AS an INFO message that includes the following MSML control syntax describing the DTMF Detect event.

```
<msml version="1.0">
   <event name="done"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:13">
      <name>dtmf.digits</name><value>2</value>
      <name>dtmf.end</name><value>dtmf.detect</value>
      <name>dtmf.last</name><value>2</value>
      <name>dtmf.len</name><value>1</value>
   </event>
</msml>
```

Step 4b

The AS sends a 200 OK response to the MS to acknowledge successful receipt of the DTMF Detect event. The 200 OK response does not includes any MSML control syntax.

Step 4c

The MS sends the AS an INFO message with the following MSML control syntax to indicate that the **Video Portal Prompt** dialog is exiting (a response to the DTMF Detect event).

```
<msml version="1.0">
   <event name="msml.dialog.exit"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:13"/>
</msml>
```

Step 4d

The AS sends a 200 OK response to the MS to acknowledge **Video Portal Prompt** dialog exit. The 200 OK response does not contain any MSML control syntax.

Step 4e

The AS sends the MS an INFO message that includes the following MSML control syntax to start the **Video Clip** (a response to the DTMF Detect event).

```
<?xml version="1.0" encoding="UTF-8" ?>
<msml version="1.0">
<dialogstart target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2"
      type="application/moml+xml">
   <play>
      <media>
         <video uri="file://./av/clip2.vid" format="video/raw:codecs=h263" />
         <audio uri="file://./av/clip2.pcm" format="audio/pcm:codecs=mulaw"
            audiosamplesize="8" audiosamplerate="8"  />
      </media>
   </play>
</dialogstart>
</msml>
```

Step 4f

The MS sends a 200 OK response to the AS to acknowledge the starting of the **Video Clip**. The 200 OK response includes the following MSML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:28</dialogid>
</msml>
```

Step 4g

The MS sends the AS an INFO message with the following MSML control syntax to indicate that the **Video Clip** dialog is exiting (a response to the DTMF Detect event).

```
<msml version="1.0">
   <event name="msml.dialog.exit"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:28"/>
</msml>
```

Step 4h

The AS sends a 200 OK response to the MS to acknowledge **Video Clip** dialog exit. The 200 OK response does not contain any MSML control syntax.

Step 4i

The AS sends the MS an INFO message to play the **Main Prompt** dialog. The INFO message includes the following MSML control syntax:

```
<?xml version="1.0" encoding="UTF-8" ?><msml version="1.0"><dialogstart
      target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2" type="application/moml+xml">
   <group topology="parallel" >
      <play>
         <media>
            <video uri="file://./av/main_menu.vid" format="video/raw:codecs=h263" />
            <audio uri="file://./av/main_menu.pcm" format="audio/pcm:codecs=mulaw"
               audiosamplesize="8" audiosamplerate="8"/>
         </media>
      </play>
      <collect iterate="forever" cleardb="true">
         <detect>
            <send target="source" event="done" namelist="dtmf.digits" />
            <send target="group" event="terminate"/>
         </detect>
      </collect>
   </group>
</dialogstart>
</msml>
```

Step 4j

The MS sends a 200 OK response to indicate success. The 200 OK response includes the following MSML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:8</dialogid>
</msml>
```

3.2.5 Record Message

Step 5a

At this point, the SIP client makes a selection that is transmitted as DTMF to the MS. The selection is to **Record Message**. The MS sends the AS an INFO message that includes the following MSML control syntax describing the DTMF Detect event.

```
<msml version="1.0">
   <event name="done"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:8">
      <name>dtmf.digits</name><value>2</value>
</event>
</msml>
```

Step 5b

The AS sends a 200 OK response to the MS to acknowledge successful receipt of the DTMF Detect event. The 200 OK response does not includes any MSML control syntax.

Step 5c

The MS sends the AS an INFO message with the following MSML control syntax to indicate that the **Main Prompt** dialog is exiting (a response to the DTMF Detect event):

```
<msml version="1.0">
   <event name="msml.dialog.exit"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:8"/>
</msml>
```

Step 5d

The AS sends a 200 OK response to the MS to acknowledge **Main Prompt** dialog exit. The 200 OK response does not contain any MSML control syntax.

Step 5e

The AS sends the MS an INFO message that includes the following MSML control syntax to start the **Record Message** dialog (a response to the DTMF Detect event).

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.0">
   <dialogstart target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2"
      type="application/moml+xml">
      <group topology="parallel">
         <record beep="true"    audiodest="file://./mytest.pcm"
               videodest="file://./mytest.vid"    format="video/raw;codecs=mulaw,h263"
               audiosamplerate="8" audiosamplesize="8">
            <recordexit>
               <send target="play" event="resume"/>
            </recordexit>
         </record>
         <play initial="suspend">
            <media>
               <audio uri="file://./mytest.pcm" format="audio/pcm;codecs=mulaw"
                     audiosamplerate="8" audiosamplesize="8" />
               <video uri="file://./mytest.vid" format="video/vid;codecs=h263"/>
            </media>
            <playexit>
               <send target="dtmf" event="terminate"/>
               <send target="record" event="terminate"/>
            </playexit>
         </play>
         <dtmf iterate="forever">
            <detect>
```

```
                <send target="record" event="terminate"/>
             </detect>
          </dtmf>
       </group>
    </dialogstart>
 </msml>
```

Step 5f

The MS sends a 200 OK response to the AS to acknowledge the starting of the **Record Message** dialog. The 200 OK response includes the following MSML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:30</dialogid>
</msml>
```

Step 5g

Once the MS is ready to record, it sends the AS an INFO message with the following MSML/MOML control syntax to indicate that the MS is waiting for an I-frame. The AS must forward this message to the remote SIP client.

```
<?xml version="1.0" encoding="utf-8" ?>
<media_control><vc_primitive><to_encoder><picture_fast_update></picture_fast_update></
 to_encoder></vc_primitive></media_control>
```

Recording starts once an I-frame is received.

If the MS does not get an I-frame within 5 seconds, another message with the same syntax is sent to the AS indicating that an I-frame timeout occurred. The AS must also forward this message to the remote SIP client. The MS will start recording without a valid I-frame.

3.2.6 Replay Main Prompt

Step 6a

At this point, when the SIP client sends any DTMF to the MS, the recording operation stops and playback begins automatically (as determined by the MSML control syntax in Step 5e). The MS also sends the AS an INFO message that includes the following MSML control syntax indicating a **Record Message** dialog exit event.

```
<msml version="1.0">
   <event name="msml.dialog.exit"
      id="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:30"/>
</msml>
```

Step 6b

The AS sends a 200 OK response to the MS to acknowledge successful receipt of the event. The 200 OK response does not includes any MSML control syntax.

Step 6c

When the playback of the recorded message is complete, the AS sends the MS an INFO message to play the **Main Prompt** dialog. The INFO message includes the following MSML control syntax:

```
<?xml version="1.0" encoding="UTF-8" ?><msml version="1.0">
<dialogstart target="conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2"
type="application/moml+xml">
   <group topology="parallel" >
      <play>
         <media>
            <video uri="file://./av/main_menu.vid" format="video/raw:codecs=h263" />
            <audio uri="file://./av/main_menu.pcm" format="audio/pcm:codecs=mulaw"
                  audiosamplesize="8" audiosamplerate="8"/>
         </media>
      </play>
      <collect iterate="forever" cleardb="true">
         <detect>
            <send target="source" event="done" namelist="dtmf.digits" />
            <send target="group" event="terminate"/>
         </detect>
      </collect>
   </group>
</dialogstart>
</msml>
```

Step 6d

The MS sends a 200 OK response to indicate success. The 200 OK response includes the following MSML/MOML control syntax:

```
<msml version="1.0">
   <result response="200"/>
   <dialogid>conn:b19d82a0-0-13c4-13cbd2-58c3964e-13cbd2/dialog:29</dialogid>
</msml>
```

3.2.7 Terminate Connections

Step 7a to 7d

These steps comprise standard SIP message exchanges for the termination of the SIP dialogs between the SIP client and the AS and between the AS and MS and the termination of the media session (RTP) between the SIP client and the MS.

Feature Support 4

This chapter describes the high-level features supported by the current version of the MSML Media Server Software, MSML and MOML protocol support, and other related topics.

4.1 Feature Highlights by Platform

Table 1 presents the high-level features and functionality supported in the current version of the MSML Media Server Software. It also lists the Dialogic® platform (software release) that supports each feature.

Features and functionality supported are being introduced in phases. This table will be updated as a feature becomes available in a particular platform (software release) or service update (SU).

Table 1. High-Level Feature Support by Platform

Feature	Dialogic® HMP Software 3.0WIN	Dialogic® HMP Software 3.1LIN	MMP for ATCA	Feature Details / Comments
Audio conferencing	S	S	S	
Audio play and record	S	S	S	
Audit package	S (SU 157)	S (SU 157)	NS	
Digit detection - inband	S	S	S	
Digit detection - RFC 2833	S	S	S	
EVRC codec	NS	NS	S	
HTTP play and record	S	S	S	See Section 5.3, “HTTP Play and Record”, on page 46.
I-frame	S	S	S	See Section 3.2.5, “Record Message”, on page 22, step 5G. I-frame timeout notification supported; I-frame on-demand not yet supported.
PSTN	S (SU 176)	S (SU 174)	NS	
SIP re-INVITE	S	S	S	See Section 5.2, “SIP re-INVITE”, on page 44
SIP session timers (keepalive)	S	S	S	See Section 5.1, “SIP Session Timer”, on page 41.
Video play and record	S (SU 176)	S (SU 157)	NS	
Legend: S = supported; NS = not supported Dialogic® HMP Software 3.0WIN = Dialogic® Host Media Processing Software Release 3.0WIN Dialogic® HMP Software 3.1LIN = Dialogic® Host Media Processing Software Release 3.1LIN MMP for ATCA = Dialogic® Multimedia Platform for AdvancedTCA				

4.2 Media Server Markup Language (MSML) Protocol Support Matrix

Table 2 describes the current level of support for MSML elements and attributes. Supported items are shown in black text; unsupported items are shown in red text. The “Comment” column indicates restrictions or limitations that the current version of the MSML Media Server Software imposes.

Note: The level of support is correlated against the MSML IETF draft, saleem-msml-00.

Table 2. MSML Protocol Support Matrix

Element Name	Attribute Name	Level of Support	Comments
<msml>	-	supported	
	version	supported	Must be “1.0”
<send>	-	supported	
	event	supported	
	target	supported	
	valuelist	supported	
	mark	supported	
<result>	-	supported	
	response	supported	
	mark	supported	
<event>	-	supported	
	name	supported	
	id	supported	
<join>	-	supported	
	id1	supported	
	id2	supported	
	mark	supported	
<modifystream>	-	supported	
	id1	supported	
	id2	supported	
	mark	supported	
<unjoin>	-	supported	
	id1	supported	
	id2	supported	
	mark	supported	
<stream>	-	supported	
	media	supported	Supports “audio” only, which is the default
	dir	supported	
	compressed	not supported	
	preferred	not supported	
	display	not supported	

Table 2. MSML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Level of Support	Comments
<gain>	-	supported	
	amt	supported	Not supported if the stream direction is from a conference ID.
	agc	supported	
	tgtlvl	not supported	
	maxgain	not supported	
<clamp>	-	supported	Limited to <modifystream> where the stream direction is to a conference ID.
	dtmf	supported	Mandatory field; can be set to “true” or “false”.
	tone	supported	Mandatory field; must always be set to “false”.
<visual>	-	not supported	
<monitor>	-	not supported	
	id1	not supported	
	id2	not supported	
	compressed	not supported	
	mark	not supported	
<createconference>	-	supported	Limited to audio conferences.
	name	supported	
	deletewhen	supported	
	term	supported	
	mark	supported	
<modifyconference>	-	supported	
	id	supported	
	mark	supported	
<destroyconference>	-	supported	
	id	supported	
	mark	supported	
<audiomix>	-	supported	
	id	supported	
<n-loudest>	-	not supported	The number of audio mix participants is set to 3 and cannot be changed.
	id	not supported	

Table 2. MSML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Level of Support	Comments
<asn>	-	supported	Minimum interval is 1 second.
	n	supported	Can be specified in seconds “s”, milliseconds “ms”, or minutes “m”. Default is “ms” if no units are specified.
<videolayout>	-	not supported	
	type	not supported	
	id	not supported	
<root>	-	not supported	
	size	not supported	
	backgroundcolor	not supported	
	backgroundimage	not supported	
<region>	-	not supported	
	id	not supported	
	left	not supported	
	top	not supported	
	relativesize	not supported	
	priority	not supported	
	title	not supported	
	titletextcolor	not supported	
	titlebackgroundcolor	not supported	
	bordercolor	not supported	
	borderwidth	not supported	
	logo	not supported	
	freeze	not supported	
	blank	not supported	
<selector>	-	not supported	
	id	not supported	
	method	not supported	
	status	not supported	
	blankothers	not supported	
	si	not supported	
	speakersees	not supported	
<reserve>	-	not supported	
	required	not supported	

Table 2. MSML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Level of Support	Comments
<resource	-	not supported	
	n	not supported	
<dialogstart>	-	supported	
	target	supported	
	src	supported	Supports the following schemes: • "file://" • "http://"
	type	supported	
	name	supported	
	mark	supported	
<dialogend>	-	supported	
	id	supported	
	mark	supported	

4.3 Media Object Markup Language (MOML) Protocol Support Matrix

Table 3 describes the current level of support for MOML elements, attributes, events and shadow variables. Supported items are shown in black text; unsupported items are shown in red text. The “Comment” column indicates restrictions or limitations that the current version of the MSML Media Server Software imposes.

Note: The level of support is correlated against the MSML IETF draft, saleem-msml-00.

Table 3. MOML Protocol Support Matrix

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<moml>				supported	
	version			supported	Must be “1.0”
	id			supported	
		terminate		supported	
<send>				supported	
	event			supported	
	target			supported	
	namelist			supported	
<exit>				not supported	
	namelist			not supported	
<disconnect>				not supported	
	namelist			not supported	
<event>				supported	
	name			supported	
	id			supported	
<group>				supported	
	topology			supported	Supports “parallel” topology only
	id			supported	
		terminate		supported	
<groupexit>				supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<play>				supported	
	id			supported	
	interval			supported	
	iterate			supported	
	initial			supported	
	maxtime			supported	
	barge			supported	No effect for video
	cleardb			supported	Supports “true” only No effect for video
	offset			supported	No effect for video
	skip			not supported	
	xml:lang			not supported	
		pause		not supported	
		resume		supported	
		forward		not supported	
		backward		not supported	
		restart		supported	
		toggle-state		not supported	
		terminate		supported	
			play.amt	supported	No support for video
			play.end	supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<audio>					
	uri			supported	Supports the following schemes: • "file://" • "http://"
	format			supported	Supports "audio/wav" and "audio/vox;codecs=*value*" only Codecs are ignored for wav format Codecs are required for vox format; valid codecs are: "mulaw", "alaw", "pcm", "dialogic_adpcm", "g726"
	audiosamplerate			supported	Ignored for wav format Required for vox format; valid values are: 6, 8, 11
	audiosamplesize			supported	Ignored for wav format Required for vox format; valid values are: 2, 4, 8
	iterate			supported	
	xml:lang			not supported	
<video>				supported	
	uri			supported	Supports the following schemes: • "file://" • "http://"
	format			supported	Supports "video/vid;codecs=h263" only
	audiosamplerate			not supported	
	audiosamplesize			not supported	
	codecconfig			not supported	
	profile			supported	No values defined
	level			supported	No values defined
	imagewidth			supported	No values defined
	imageheight			supported	No values defined
	maxbitrate			supported	No values defined
	framerate			supported	No values defined
	iterate			supported	
<media>				supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<var>				not supported	
	type			not supported	
	subtype			not supported	
	value			not supported	
	xml:lang			not supported	
<playexit>				supported	
<dtmfgen>				supported	
	id			supported	
	digits			supported	
	level			supported	
	dur			supported	
	interval			supported	
		terminate		supported	
			dtmfgen.end	supported	
<dtmfgenexit>				supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<record>				supported	
	id			supported	
	append			not supported	
	dest			supported	Uses dx_ device Supports the following schemes: • “file://” • “http://”
	audiodest			supported	Uses mm_ device Supports the following schemes: • “file://” • “http://”
	videodest			supported	Uses mm_ device Supports the following schemes: • “file://” • “http://”
	format			supported	
	codeconfig			not supported	
	audiosamplerate			supported	Valid values are: 6, 8, 11
	audiosamplesize			supported	Valid values are: 2, 4, 8
	profile			supported	No values defined
	level			supported	No values defined
	imagewidth			supported	No values defined
	imageheight			supported	No values defined
	maxbitrate			supported	No values defined
	framerate			supported	No values defined
	initial			supported	
	maxtime			supported	
	prespeech			not supported	
	postspeech			supported	
	termkey			supported	No effect for video
	beep			supported	Proprietary implementation
		pause		not supported	
		resume		supported	
		toggle-state		not supported	
		terminate		supported	
		terminate.cancelled		not supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<recordexit>				supported	
<dtmf>, <collect>				supported	
	id			supported	
	cleardb			supported	Supports “true” only
	fdt			supported	
	idt			supported	
	starttimer			supported	
	iterate			supported	
		starttimer		supported	
		terminate		supported	
			dtmf.digits	supported	
			dtmf.len	supported	
			dtmf.last	supported	
			dtmf.end	supported	
<pattern>	-			supported	
	digits			supported	
	format			supported	Supports the “moml+digits” format only; the “mgcp” and “megaco” formats are not supported
	iterate			supported	
<detect>				supported	
<noinput>				supported	
	iterate			supported	
<nomatch>				supported	
	iterate			supported	
<dtmfexit>				supported	
<vad>				not supported	
	starttimer			not supported	
		starttimer		not supported	
		terminate		not supported	
<voice>, <silence>				not supported	
	len			not supported	
<tvoice>, <tsilence>				not supported	
	sen			not supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<gain>				supported	
	incr			supported	
	amt			supported	Valid values are in the range: -10 to +10
		mute		not supported	
		unmute		not supported	
		reset		supported	
		louder		supported	
		softer		supported	
		amt		supported	
<agc>				not supported	
	tgtlvl			not supported	
	maxgain			not supported	
		mute		not supported	
		unmute		not supported	
<gate>				not supported	
	initial			not supported	
		mute		not supported	
		unmute		not supported	
<clamp>				not supported	
<relay>				not supported	
<speech>				not supported	
	noint			not supported	
	norect			not supported	
	spcmplt			not supported	
	spincmplt			not supported	
	confidence			not supported	
	sens			not supported	
	starttimer			not supported	
	iterate			not supported	
		sens		not supported	
		starttimer		not supported	
		terminate		not supported	
			speech.end	not supported	
			speech.results	not supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<grammar>				not supported	
	uri			not supported	
	iterate			not supported	
<match>				not supported	
<noinput>				not supported	
	iterate			not supported	
<nomatch>				not supported	
	iterate			not supported	
<speechexit>				not supported	
<tts>				not supported	
	uri			not supported	
	iterate			not supported	
	xml:lang			not supported	
<faxdetect>				not supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<faxsend>				not supported	
	lclid			not supported	
	minspeed			not supported	
	maxspeed			not supported	
	ecm			not supported	
		terminate		not supported	
			fax.rmtid	not supported	
			fax.rate	not supported	
			fax.resolution	not supported	
			fax.pagesize	not supported	
			fax.encoding	not supported	
			fax.ecm	not supported	
			fax.pagebadlines	not supported	
			fax.objbadlines	not supported	
			fax.opbadlines	not supported	
			fax.objuri	not supported	
			fax.resendcount	not supported	
			fax.totalpages	not supported	
			fax.totalobjects	not supported	
			fax.duration	not supported	
			fax.result	not supported	
<sendobj>				not supported	
	objuri			not supported	
	startpage			not supported	
	pagecount			not supported	
<hdrfooter>				not supported	
	type			not supported	
	style			not supported	
<format>				not supported	
<rxpoll>				not supported	
	rmtid			not supported	
<faxstart>				not supported	
<faxnegotiate>				not supported	
<faxpagedone>				not supported	

Table 3. MOML Protocol Support Matrix (Continued)

Element Name	Attribute Name	Event Name	Shadow Variable Name	Level of Support	Comment
<faxobjectdone>				not supported	
<faxopcomplete>				not supported	
<faxpollstarted>				not supported	
<faxrcv>				not supported	
	lclid			not supported	
	ecm			not supported	
		terminate		not supported	
			fax.rmtid	not supported	
			fax.rate	not supported	
			fax. resolution	not supported	
			fax. pagesize	not supported	
			fax.encoding	not supported	
			fax.ecm	not supported	
			fax.pagebadlines	not supported	
			fax.objbadlines	not supported	
			fax. opbadlines	not supported	
			fax.objuri	not supported	
			fax.resendcount	not supported	
			fax.totalpages	not supported	
			fax.totalobjects	not supported	
			fax.duration	not supported	
			fax.result	not supported	
<rcvobj>				not supported	
	objuri			not supported	
	maxpages			not supported	
<txpoll>				not supported	

Feature Details 5

This chapter describes the features supported by the current version of the MSML Media Server Software in detail and provides information on how to use these features. Topics include:

Note: A feature may not be supported on all platforms. For more information, see Table 1, “High-Level Feature Support by Platform”, on page 26 in Chapter 4, “Feature Support” .

5.1 SIP Session Timer

The session timer feature provides a keepalive mechanism for the Session Initiation Protocol (SIP) to determine whether a session is still active using an extension defined in RFC 4028.

5.1.1 Feature Description

Session timer can be enabled by the application server on a per call basis through a SIP INVITE request to the media server. The support is disabled in the media server by default. The application server can also set the timer refresh interval on a per call basis.

For every SIP session, the session timer is active if requested by the application server. If the session timer is active, the application server sends a refresh message to the media server at regular intervals. The media server must respond to the refresh message, which is a SIP UPDATE request. The refresh response is a 200 OK.

If the media server does not receive the refresh message during the refresh interval, the associated SIP session is cleared by the media server. Any active dialog associated with the SIP session is terminated.

If the application server does not receive the refresh response from the media server during the refresh interval, the associated SIP session is cleared by the application server.

5.1.2 Enabling SIP Session Timer

The session timer support in the media server is disabled by default. It is enabled when the application server requests session timers through the SIP INVITE request.

This section illustrates a sample scenario where the application server (AS) requests session timer support from the media server (MS).

AS: SIP INVITE request for session timer support

The application server sends a SIP INVITE request to the media server requesting session timer support (see text in bold).

```
INVITE sip:192.168.1.227:5060 SIP/2.0
From: <sip:192.168.1.210:5060>;tag=2cd4db0-0-13c4-4086a-2f8ff578-4086a
To: <sip:192.168.1.227:5060>
Call-ID: 2ce18c8-0-13c4-4086a-4de0d2d3-4086a@192.168.1.210
CSeq: 1 INVITE
Via: SIP/2.0/UDP 192.168.100.160:5060;branch= 4086a-fc0df70-31ac7aa1
Max-Forwards: 70
Supported: timer
Contact: <sip:192.168.1.210:5060>
Session-Expires: 90;refresher=uac
Min-SE: 90
Allow: INVITE, CANCEL, ACK, BYE, OPTIONS, UPDATE
Content-Type: application/SDP
Content-Length: 239

v=0
o=- 1032127810 1032127810 IN IP4 192.168.1.215
s=Polycom IP Phone
c=IN IP4 192.168.1.215
t=0 0
m=audio 2240 RTP/AVP 0 8 18 101
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:18 G729/8000
a=rtpmap:101 telephone-event/8000
```

MS: SIP response to request for session timer support

The media server responds with a 200 OK message and agrees to use session timers (see text in bold).

```
SIP/2.0 200 OK
From: <sip:192.168.1.210:5060>;tag=2cd4db0-0-13c4-4086a-2f8ff578-4086a
To: <sip:192.168.1.227:5060>;tag=31d86c8-0-13c4-42a04-4f95866a-42a04
Call-ID: 2ce18c8-0-13c4-4086a-4de0d2d3-4086a@192.168.1.210
CSeq: 1 INVITE
Via: SIP/2.0/UDP 192.168.100.160:5060;received=192.168.1.210;branch=
       4086a-fc0df70-31ac7aa1
Supported: timer
Require: timer
Contact: <sip:192.168.1.227:5060>
Session-Expires: 90;refresher=uac
Content-Type: application/SDP
Content-Length: 268
v=0
o=MediaServer 1180967421 1180967421 IN IP4 192.168.1.227
s=-
i=Media Server Version 1,0,0,75
t=0 0
m=audio 49188 RTP/AVP 0 101
c=IN IP4 192.168.1.227
a=rtpmap:0 PCMU/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:0 ptime=20
a=fmtp:101 0-15
a=sendrecv
```

At this point, the call referenced above is active and session timer support is active.

AS: SIP UPDATE request

Once the SIP session is established with session timers enabled, the application server sends a SIP UPDATE request to the media server at regular intervals, as specified in the initial INVITE. Once the media server receives the SIP UPDATE, it responds to the session timer refresh request with 200 OK.

```
UPDATE sip:192.168.1.227:5060 SIP/2.0
From: <sip:192.168.1.210:5060>;tag=2cd4db0-0-13c4-4086a-2f8ff578-4086a
To: <sip:192.168.1.227:5060>;tag=31d86c8-0-13c4-42a04-4f95866a-42a04
Call-ID: 2ce18c8-0-13c4-4086a-4de0d2d3-4086a@192.168.1.210
CSeq: 38 UPDATE
Via: SIP/2.0/UDP 192.168.100.160:5060;branch= 40cc0-fd1d1a6-29630a07
Max-Forwards: 70
Supported: timer
Contact: <sip:192.168.1.210:5060>
Session-Expires: 90;refresher=uac
Min-SE: 60
Allow: INVITE, CANCEL, ACK, BYE, OPTIONS, UPDATE
Content-Length: 0
```

MS: SIP response to SIP UPDATE request

The media server responds to the SIP UPDATE request from the application server with 200 OK.

If session timer support is enabled, the call remains active as long as the application server sends the SIP UPDATE request and the media server responds with 200 OK.

If the application server does not send the SIP UPDATE request after the session timer expires, or the media server does not respond with 200 OK, then the call is automatically terminated.

```
SIP/2.0 200 OK
From: <sip:192.168.1.210:5060>;tag=2cd4db0-0-13c4-4086a-2f8ff578-4086a
To: <sip:192.168.1.227:5060>;tag=31d86c8-0-13c4-42a04-4f95866a-42a04
Call-ID: 2ce18c8-0-13c4-4086a-4de0d2d3-4086a@192.168.1.210
CSeq: 38 UPDATE
Via: SIP/2.0/UDP 192.168.100.160:5060;received=192.168.1.210;branch=
      40cc0-fd1d1a6-29630a07
Supported: timer
Require: timer
Contact: <sip:192.168.1.227:5060>
Session-Expires: 90;refresher=uac
Content-Length: 0
```

5.1.3 Using SIP Session Timer with Dialogic® HMP Software

This section applies only to application servers that use Dialogic® Host Media Processing (HMP) Software as the call control mechanism.

To activate the session timer support in the media server, the application server must request session timer support in the SIP INVITE request.

The following procedure outlines the steps required to request session timer support from the media server using the Dialogic® Global Call API library in Dialogic® HMP Software.

1. Enable the session timer support on the virtual board on the application server.

   ```
   m_virtBoard[0].E_SIP_SessionTimer_Enabled= ENUM_Enabled;
   ```

2. Set the default session timer refresh interval and minimum refresh interval as appropriate. The default session expires timer is set to 90 seconds below. If not set, it defaults to 1800 seconds.

   ```
   m_virtBoard[0].SIP_SessionTimer_SessionExpires= 90;

   m_virtBoard[0].SIP_SessionTimer_MinSE= 90;
   ```

3. Set the following IPPARM values on each SIP device.

```
gcParmBlk = NULL;
gc_util_insert_parm_val(&gcParmBlk,
        IPSET_SIP_SESSION_TIMER,
        IPPARM_REFRESH_METHOD,
        sizeof(UINT32),
        IP_REFRESH_UPDATE);
gc_util_insert_parm_val(&gcParmBlk,
        IPSET_SIP_SESSION_TIMER,
        IPPARM_REFRESHER_PREFERENCE,
        sizeof(UINT32),
        IP_REFRESHER_LOCAL);
gc_util_insert_parm_val(&gcParmBlk,
        IPSET_SIP_SESSION_TIMER,
        IPPARM_REFRESH_WITHOUT_REMOTE_SUPPORT,
        sizeof(UINT32),
        IP_REFRESH_WITHOUT_REMOTE_SUPPORT_DISABLE);
gc_util_insert_parm_val(&gcParmBlk,
        IPSET_SIP_SESSION_TIMER,
        IPPARM_REFRESH_WITHOUT_PREFERENCE,
        sizeof(UINT32),
        IP_REFRESH_WITHOUT_PREFERENCE_DISABLE);
gc_SetUserInfo(GCTGT_GCLIB_CHAN, m_DeviceHandle, gcParmBlk,
              GC_ALLCALLS);
gc_util_delete_parm_blk(gcParmBlk);
```

Note that the application server itself is not responsible for sending the session timer refresh requests. The underlying Dialogic® HMP Software manages interactions related to session timer requests and responses.

For more information about the Dialogic® Global Call API functions and parameters used for SIP session timer, see the SIP Session Timer topic in the Release Update for your corresponding software release. For more information about Global Call API for IP, see the Dialogic® Global Call IP Technology Guide.

5.2 SIP re-INVITE

SIP re-INVITE support is provided to handle media flow direction control changes, media capabilities renegotiation, and outbound media destination IP address changes.

5.2.1 Feature Description

The re-INVITE method is a general purpose mechanism that can be used to modify or update most of the properties of a dialog (the most notable exceptions being the header fields that are used to identify the message as a subsequent INVITE rather than a new INVITE) or the associated media session.

This initial implementation of re-INVITE does not support the following:

- Requests sent from the media server (MS). The media server will only receive re-INVITE requests and respond to them.
- Hold or Retrieve when the “inactive” attribute is used.
- Switching to Fax.

5.2.2 Media Stream Direction Support

This feature supports both full-duplex and half-duplex media streams. The direction of the media stream is based on the direction attribute supplied in the Session Description Protocol (SDP) using one of the following direction attributes, “sendrecv,” “sendonly,” or “recvonly.”

- If no direction attribute is supplied and a valid connection address “c=” is supplied, a full duplex media stream is established.
- If a connection address “c=” is set to 0.0.0.0, then a half duplex receive only media stream is established.
- The re-INVITE is rejected if the connection address “c=” is set to 0.0.0.0 and the direction attribute is “sendrecv” or “recvonly”, or if the direction attribute is set to “inactive”.

Support is provided for receiving an initial inbound INVITE with or without a valid SDP. In the case where the INVITE received does not contain an SDP, the media server (MS) will respond with an SDP offer in the 200 OK. The media flow direction is based on the answer SDP. When the inbound INVITE contains an SDP, the MS will respond with a valid SDP answer in the 200 OK and the media flow direction will be based on the offer SDP.

When an initial inbound INVITE is received with a valid SDP with a connection address “c=” value of 0.0.0.0, the MS responds with a valid SDP answer in the 200 OK. Media flow will be started as receive only.

A re-INVITE can be received any time after the call is established. Multiple re-INVITE requests can be received on the same SIP session, but prior requests must be processed and responded to before a new request can be received.

The media stream remains active based on the previous media stream capabilities and direction. A re-INVITE request with an invalid SDP will be rejected, but this has no effect on the current media stream.

The answer or offer SDP contains a valid connection address “c=”. If the connection address “c=” equals 0.0.0.0 and no direction attribute is set, then a direction attribute of “sendonly” is assumed. If a direction attribute other than “sendonly” is set, the re-INVITE is rejected. The answer or offer SDP direction attribute cannot be set to “inactive”.

5.3 HTTP Play and Record

The application server has the ability to store media recordings directly to an HTTP server. The application server can also play a recording and/or retrieve MSML/MOML dialog source directly from an HTTP server.

5.3.1 Feature Description

HTTP support enables the retrieval of MOML scripts directly from a web server for execution. It also allows the storage and retrieval of audio and video recordings to and from the HTTP server.

5.3.2 Requirements for HTTP Support

The MSML Media Server Software uses the HTTP GET and HTTP PUT commands to retrieve and store files respectively. When using MSML attributes that specify the “http://” scheme, it is important that the HTTP server support the HTTP GET and HTTP PUT commands. Typically, HTTP servers support the HTTP GET command, but when receiving HTTP PUT commands, some servers require server-side scripts to actually store files.

5.3.3 Dialog Execution

An external MOML dialog/script can be retrieved and executed from an HTTP server. Dialogs are a class of objects that represent automated participants. Dialogs are created and destroyed through MSML.

The “http://” scheme for the src attribute of an MSML <dialogstart> is supported. MOML dialog execution commences after the entire MOML body has been retrieved from the HTTP server. The HTTP GET functionality is used to retrieve the information from the web server.

5.3.4 Audio and Video Playback

Both audio and video files are retrievable from the HTTP server.

An audio file/prompt can start playing before the entire file has been downloaded from the HTTP server. Currently, audio playback support is only available when playing a VOX or WAV file using a DX device via user I/O.

If an audio file is being played in conjunction with a video file, playback will not start until the entire audio and video files are downloaded. The "http://" scheme is supported in the uri attribute of the MOML <audio> and <video> elements. The uri attribute identifies the location of the audio or video file. The HTTP GET functionality will be used to retrieve the information from the web server. Currently, only the Dialogic proprietary video format is supported.

5.3.5 Audio and Video Recording

MOML supports storing audio and video files to an HTTP server. Both are stored locally and uploaded to the HTTP server immediately after the recording has completed.

New audio files are stored in the location identified in the “http://” scheme for the dest attribute of the MOML <record> element. The audio portion of an audio / video recording is stored in the location specified at “http://” scheme for the audiodest attribute, while the location of the new video file is stored in the location identified at “http://” scheme for the videodest attribute of the MOML <record> element. The HTTP PUT functionality is used to send the information to the web server.

Deviations from IETF Draft 6

The version of the MSML Media Server Software described in this manual is based on the IETF Draft, saleem-msml-00.

The following is a list of deviations from the IETF draft:

- Nested groups are not supported.
- Only the "parallel" **topology** for <**group**> elements is supported.
- The MOML draft calls for the <**play**> element to be a child of a <**record**> or <**dtmf**> element. This is not supported.
- Wildcard IDs for the <**join**>, <**modifystream**> and <**unjoin**> elements are not supported.
- The <**video**> element is not supported as a direct child of the <**play**> element. The <**audio**> and <**video**> elements must be child elements of the <**media**> element to play an audio-visual recording.
- The **barge**, **cleardb**, and **offset** attributes of the <**play**> element have no effect when playing a video.
- The **play.amt** shadow variable of the <**play**> element has no effect for video.
- The **record.len** shadow variable of the <**record**> element has no effect for video.
- When recording an audio item, the **dest** attribute must be used.
- When recording an audio-visual item, the **audiodest** and **videodest** attributes must be used.
- Audio-visual recordings are currently recorded into two separate files.
- Audio-visual playback is supported via two separate files and must be defined in an <**audio**> and <**video**> element.
- The **format** attribute of the <**pattern**> element supports the "moml+digits" format only. The "mgcp" and "megaco" formats are not supported.
- The **amt** attribute of the <**gain**> element is not supported if the stream direction is from a conference ID.
- The <**createconference**> primitive only supports audio conferences.

Diagnostics 7

This chapter provides an overview of the diagnostic capabilities available in the MSML Media Server Software. Topics include:

7.1 Overview

The media server is integrated with the RTF logging services used by other host libraries. The RTF XML configuration file, located in the *cfg* subdirectory, is used to set the logging level that defines what content is written to the RTF log file. The media server logging capabilities are described in the subsections following.

7.2 Logging Configuration

Different media server client objects can be assigned different logging levels. The following topics provide more detail:

- Configuration File Format
- Logging Labels
- Client Categories

7.2.1 Configuration File Format

The media server section of the RTF configuration file is given below. Detailed information about the logging labels and the clients that can be configured for logging is given in the sections following.

```
<!-- IP Media Server-->
<Module name="media_svr" state = "1">
   <MLabel  name="WARN" state = "0"/>
   <MLabel  name="INFO" state = "0"/>
   <MLabel  name="FUNC" state = "0"/>

        <!-- Media Server objects-->
   <MClient name="RESOURCE" state = "0"/>
   <MClient name="SIP_BOARD_RES" state = "0"/>
   <MClient name="SIP_RES" state = "0"/>
   <MClient name="IPM_RES" state = "0"/>
   <MClient name="DX_RES" state = "0"/>
   <MClient name="MM_RES" state = "0"/>
   <MClient name="CNF_RES" state = "0"/>
   <MClient name="CNF_PARTY_RES" state = "0"/>
   <MClient name="CNF_BOARD_RES" state = "0"/>
   <MClient name="FX_RES" state = "0"/>
   <MClient name="DT_RES" state = "0"/>
   <MClient name="MSML" state = "0"/>
   <MClient name="MOML" state = "0"/>
   <MClient name="NETWORK_CONN" state = "0"/>
   <MClient name="TDM_STREAM" state = "0"/>
   <MClient name="DEV_STREAM" state = "0"/>
   <MClient name="TRANSACTION" state = "0"/>
   <MClient name="MSML_TRANSACTION" state = "0"/>
   <MClient name="MOML_TRANSACTION" state = "0"/>
   <MClient name="XML_PARSER" state = "0"/>
   <MClient name="HTTP" state = "0"/>
   <MClient name="SOCKET" state = "0"/>
   <MClient name="EVENT_MGR" state = "0"/>
   <MClient name="TRANSACTION_MGR" state = "0"/>
   <MClient name="NETWORK_CONN_MGR" state = "0"/>
   <MClient name="SESSION_MGR" state = "0"/>
   <MClient name="MEDIA_STREAM_MGR" state = "0"/>
   <MClient name="RESOURCE_MGR" state = "0"/>
   <MClient name="THREAD_MGR" state = "0"/>
   <MClient name="CNF_MGR" state = "0"/>
   <MClient name="THREAD" state = "0"/>
   <MClient name="TIMER" state = "0"/>
   <MClient name="MEDIA_CODER" state="0" />
</Module>
```

7.2.2 Logging Labels

To enable media server logging in the RTF configuration file:

```
<Module name="media_svr" state = "1">
```

To disable all media server logging:

```
<Module name="media_svr" state = "0">
```

Note: Special startup/shutdown configuration information is always logged.

The logging labels that can be specified for the various media server client objects are as follows:

ERROR - Default
: Log unexpected conditions or events that cause a related action to fail. This is a global label which is set in the beginning of the RTF XML file.

WARN
: Log unexpected conditions or events that should not cause a related action to fail.

INFO
: Log additional information such as state changes, events, and function parameters.

FUNC
: Logs function entry and exit.

Note: Logging labels are independent of each other. For example, enabling FUNC and disabling INFO logs function entry and exit, but not additional information.

7.2.3 Client Categories

The various clients that can be enabled are listed below. Each client can be set to 0 (disable) or 1 (enable). If enabled, the logging labels determine the detail of the log content. The clients can be categorized as follows:

SIP Call Control

Provides the call control implementation that allows call establishment with application servers:

- SIP_BOARD_RES – ipt virtual board abstraction
- SIP_RES – ipt channel
- NETWORK_CONN – network connection object that uses a SIP resource and conditionally an ipm resource to establish a call; also responsible for processing SDP information
- NETWORK_CONN_MGR – manager that creates, initializes and destroys network connections

Resources

Abstractions of the media resources provided by Dialogic:

- RESOURCE – base object from which all other resources are derived
- IPM_RES – ipm resource
- DX_RES – dxx resource
- MM_RES – mm resource
- CNF_RES – cnf resource
- CNF_BOARD_RES – cnf virtual board resource
- CNF_PARTY_RES – cnf party resource
- CNF_MNR – cnf manager, responsible for mapping conference names to resources and providing bridge support
- FX_RES – fax object; currently not supported
- RESOURCE_MGR – creates, allocates, deallocates and destroys all resources

MSML MOML

Objects responsible for processing MSML specific requests:

- MSML – MSML object
- MOML – MOML object

Connections

Objects responsible for creating, modifying, and destroying resource connections including gain and AGC:

- TDM_MEDIA_STREAM – implements TDM listen/unlisten functionality
- DEV_MEDIA_STREAM – implements dev_Connect/dev_Disconnect functionality
- MEDIA_STREAM_MGR – manages DEV and TDM streams

Transactions

Requests sent to the media server to perform media operations:

- TRANSACTION – base object from which all other transactions are derived
- MSML_TRANSACTION – MSML transactions
- MOML_TRANSACTION – not currently used
- TRANSACTION_MGR – allocates and deallocates transactions

Utilities

Objects that provide internal services required by clients:

- TIMER – provides timer callback trigger

- THREAD – provides a separate thread context to execute some action
- THREAD_MGR – allocates/deallocates threads
- HTTP – read from or write to remote files
- SOCKET – read from or write to TCP/IP socket; currently not supported

XML Parsing

Objects for XML parsing:

- XML_PARSER – XML parser

Miscellaneous

Other objects:

- EVENT_MGR – Standard Runtime Library (SRL) event manager; dispatches SRL events to registered clients
- SESSION_MGR – creates and destroys all other managers

7.3 Log File Format

The following is a snippet from a sample log file.

```
. . .

06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! ====> CMSML::ValidateScript()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::ValidateScript - Dumping
script body
                                                              <?xml version="1.0" encoding="UTF-8" ?>
                                                              <msml version="1.0">
                                                              <join id1="conn:51f0188-0-13d8-64325-7d02fa29-64325" id2="conn:51f0698-0-13d8-
64334-29ea58e0-64334" />
                                                              </msml>
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! <==== CMSML::ValidateScript()
returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::EvProcessScript - Validation
successful.  Processing initiated
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::EvProcessScript - Mandatory
attribute version = 1.0
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! ====> CMSML::ProcessElement()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::ProcessElement - Processing
new MSML element. Element = join
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! ====> CMSML::ProcessEvent(), event =
1
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::ProcessEvent - Processing
Event EV_Join in State MSML_Processing
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! ====> CMSML::SetState()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !              MSML !    1 ! CMSML::SetState - Transitioning from
MSML_Processing to MSML_Connecting
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! <==== CMSML::SetState()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !              MSML !    1 ! ====> CMSML::EvJoin()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====> CMSMLJoin::Parse()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====> CMSMLJoin::Reset()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! <==== CMSMLJoin::Reset() returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !          MSMLJoin !    1 ! CMSMLJoin::Parse - Processing MSML
element <join>
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====>
CMxMLElement::LoadMandatoryAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !          MSMLJoin !    1 ! CMxMLElement::LoadMandatoryAttribute
- Mandatory attribute id1 = conn:51f0188-0-13d8-64325-7d02fa29-64325
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! <====
CMxMLElement::LoadMandatoryAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====>
CMxMLElement::LoadMandatoryAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !          MSMLJoin !    1 ! CMxMLElement::LoadMandatoryAttribute
- Mandatory attribute id2 = conn:51f0698-0-13d8-64334-29ea58e0-64334
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! <====
CMxMLElement::LoadMandatoryAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====>
CMxMLElement::LoadOptionalAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !          MSMLJoin !    1 ! CMxMLElement::LoadOptionalAttribute
- Optional attribute mark = null
```

```
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! <====
CMxMLElement::LoadOptionalAttribute()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! <==== CMSMLJoin::Parse() returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====> CMSMLJoin::Execute()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !              MSML !          MSMLJoin !    1 ! ====> CMSMLJoin::CreateStreams()

06/19/2006 09:12:23.035   1300   740 media_svr    INFO !              MSML !          MSMLJoin !    1 ! CMSMLJoin::CreateStreams -
Initiating connection between conn:51f0188-0-13d8-64325-7d02fa29-64325 and conn:51f0698-0-13d8-64334-29ea58e0-64334 via MediaStreamMgr
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> CreateStream()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! Create 2 streams,
IClientNotify=0x5356284
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> SortStreams()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! Sort 2 SMediaStreams
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <=== SortStreams(): return E_SUCCESS
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> AreStreamsActive()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! Check if 2 TDM streams and 0 DEV
streams are active
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <=== AreStreamsActive(): returns
false
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> AllocateCxTransaction()
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <==== AllocateCxTransaction()
returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> AllocateCTDMStream()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! AllocateCTDMStream() return
CMediaStream with Id = 20
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <==== AllocateCTDMStream() returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! Create TDM Stream, TxResource =
ipmB1C49,  RxResource = ipmB1C47
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ===> AllocateCTDMStream()
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! AllocateCTDMStream() return
CMediaStream with Id = 19
06/19/2006 09:12:23.035   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <==== AllocateCTDMStream() returns 0
06/19/2006 09:12:23.035   1300   740 media_svr    INFO !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! Create TDM Stream, TxResource =
ipmB1C47,  RxResource = ipmB1C49
06/19/2006 09:12:23.115   1300   2260 media_svr   FUNC !       NETWORK_CONN ! CNetworkConnection !    3 ! ====> Notify()
06/19/2006 09:12:23.115   1300   2260 media_svr   INFO !       NETWORK_CONN ! CNetworkConnection !    3 ! Notify - IPMEV_LISTEN Event  -
Device = ipmB1C47
06/19/2006 09:12:23.115   1300   2260 media_svr   FUNC !       NETWORK_CONN ! CNetworkConnection !    3 ! <==== Notify() returns 8.
06/19/2006 09:12:23.115   1300   2260 media_svr   FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! ====> Notify()
06/19/2006 09:12:23.115   1300   740 media_svr    FUNC !   MEDIA_STREAM_MGR !    CMediaStreamMgr !    1 ! <=== CreateStream(): returns
E_SUCCESS
. . .
```

Media Server Markup Language (MSML) Overview A

This chapter provides a brief overview of the Media Server Markup Language (MSML). The descriptions are based on *Media Server Markup Language (MSML)*, Internet IETF Draft, saleem-msml-00, which was current during the development of the version of MSML Media Server Software described in this manual. Topics include:

A.1 Introduction

MSML is used to control and invoke different types of services on IP media servers. Clients can use MSML to define how multimedia sessions interact on a media server (MS) and to apply services to individual users or groups of users. MSML also can be used to control MS conferencing features such as video layout and audio mixing, create sidebar conferences or personal mixes, and set the properties of media streams. In addition, clients can use MSML with other languages such as the Media Objects Markup Language (MOML) to interact with individual users or groups of conference participants.

Note: The current implementation of the MSML Media Server Software does not support all the capabilities of MSML.

A.2 MSML Elements

MSML commands are sent from a client to the MS via SIP messages (most notably the INFO message). The body of the SIP message contains the XML control syntax.

A.2.1 <msml>

The root XML element of MSML is **<msml>**. It defines the set of operations that form a single MSML transaction.

Results or events returned to the client are also enclosed in the <**msml**> element.

A.2.2 <send>

The <**send**> element is used by a client to send an event to the MS.

A.2.3 <result>

The <**result**> element is used by the MS to report the results of an MSML transaction requested by a client.

A.2.4 <event>

The <**event**> element is used by the MS to notify a client of an event.

A.3 Stream Manipulation Elements

The following subsections describe the elements that establish, modify, and remove streams.

Note: The <**monitor**> element described in MSML IETF draft, version -06 is **not** currently supported.

A.3.1 <join>

The <**join**> element is used to create one or more streams between two independent objects. Streams may be audio or video and may be unidirectional or bidirectional.

A.3.2 <modifystream>

The <**modifystream**> element is used to change the properties of a stream. The <**modifystream**> element can have different properties such as the gain for an audio stream or a visual label for a video stream.

A.3.3 <unjoin>

The <**unjoin**> element is used to remove one or more existing media streams between two objects.

A.4 Conference Elements

The following subsections describe the elements that establish, modify, and destroy conferences.

A.4.1 <createconference>

The <**createconference**> element is used to create a conference. The MS assigns a conference name if the name attribute is not included.

Note: Only audio conferences are currently supported.

A.4.2 <modifyconference>

The <**modifyconference**> element is used to change the properties of a conference, such as the active talker interval.

A.4.3 <destroyconference>

The <**destroyconference**> element is used to delete an existing conference.

A.5 Dialog Elements

Dialogs are used for interaction with a user.

A.5.1 <dialogstart >

The <**dialogstart**> element is used to instantiate a media dialog on connections or conferences.

A.5.2 <dialogend >

The <**dialogend**> element is used to terminate a dialog created through <**dialogstart**>.

A.6 Receiving Events from a Client

Events are received from clients via SIP INFO messages. Events are used to affect the behavior of different objects within the MS. The client includes the <**send**> element within the <**msml**> root element. The <**send**> element identifies the event to process.

A.7 Sending Events and Transaction Results to a Client

A.7.1 Transaction Results

The <**result**> element is used to report the results of an MSML transaction. The <**result**> element is included in the final response to the SIP INFO message that initiated the transaction.

A.7.2 Events

The <**event**> element is used to notify the MS client of an event. Events are sent to clients via SIP INFO messages.

A.8 Media Server Object Model

Media server objects represent entities that source, sink, or modify media streams. A media stream is a unidirectional or bidirectional media flow between objects in a MS.

The MS object classes are:

- Network Connections (conn)
- Conference (conf)
- Dialog (dialog)
- Operator (oper)

A.8.1 Network Connections (conn)

Definition

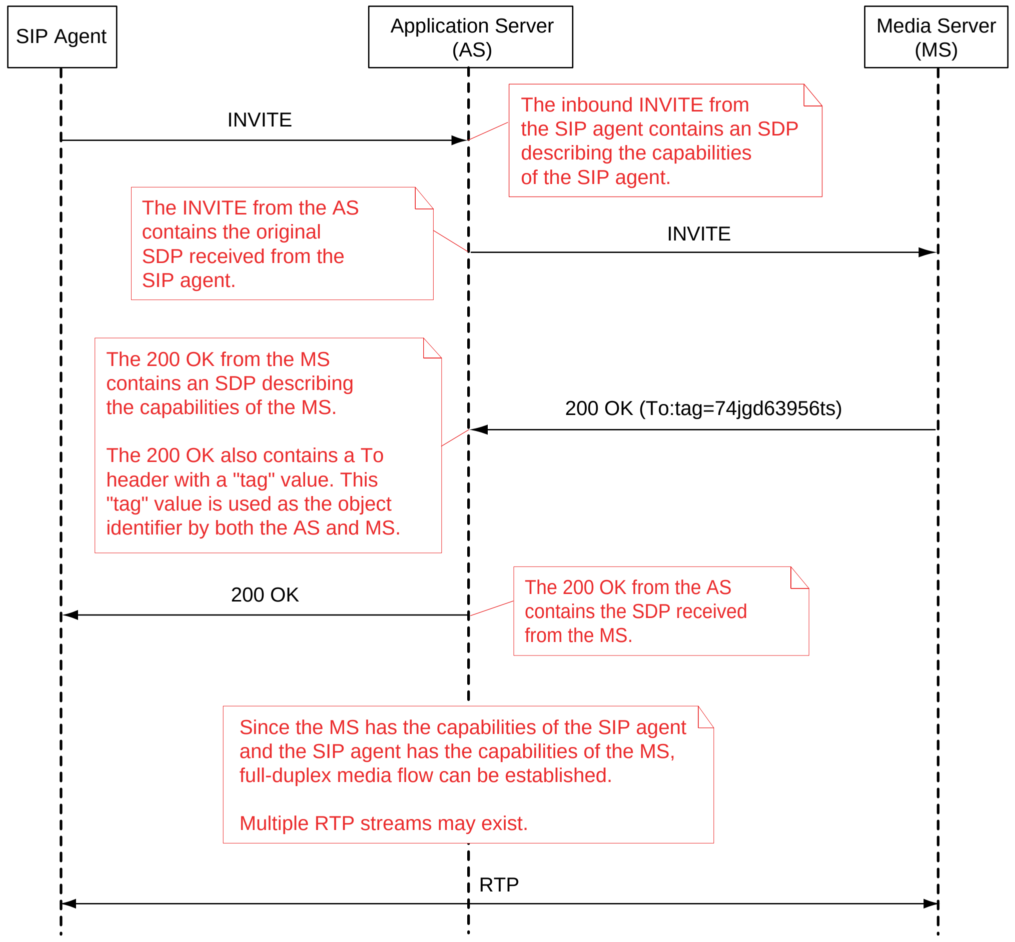
A Network Connection is an object or class defined in the MSML specification. Network Connection is an abstraction for the media processing resources involved in terminating the RTP session(s) of a call. For audio services, a connection instance presents a full-duplex audio stream interface within a MS. Multimedia connection objects have multiple media streams of different media types, each corresponding to an RTP session. MSML Network Connection instances are instantiated through SIP.

Object Creation

Unlike other MSML objects that are created using MSML commands/elements, Network Connection objects are not created using MSML commands/elements. Network Connections are created when media sessions get established through SIP call control. The connection identifier is not assigned by the AS. It is assigned by the MS and is returned to the AS in the response to the initial INVITE received from the AS. Specifically, this is the “tag” value included in the “To” header in the response. The format of the connection identifier is “conn:<tag value>”.

Figure 3 shows the interactions between the MS and the AS to create a Network Connection and establish an object identifier.

Figure 3. Network Connection Creation


SIP Agent
Application Server (AS)
Media Server (MS)
INVITE
The inbound INVITE from the SIP agent contains an SDP describing the capabilities of the SIP agent.
The INVITE from the AS contains the original SDP received from the SIP agent.
INVITE
The 200 OK from the MS contains an SDP describing the capabilities of the MS.
The 200 OK also contains a To header with a "tag" value. This "tag" value is used as the object identifier by both the AS and MS.
200 OK (To:tag=74jgd63956ts)
200 OK
The 200 OK from the AS contains the SDP received from the MS.
Since the MS has the capabilities of the SIP agent and the SIP agent has the capabilities of the MS, full-duplex media flow can be established.
Multiple RTP streams may exist.
RTP

Note: In Figure 3, the identifier used by the MS and AS to reference the network connection is “conn:74jgd63956ts”.

A.8.2 Conference (conf)

Definition

A Conference is an object or class defined in the MSML specification that allows for audio/video mixing and other advanced conferencing services. A conference represents the media resources and state information required for a single logical mix of each media type in the conference (for example, audio and video). 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.

Object Creation

Conferences are created using the <**createconference**> MSML command. The conference name can be assigned by the AS or, if the AS does not specify a name, the MS assigns one. Both the AS and the MS reference to the conference using the name as the ID, conf="name".

The AS can request that a MS 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 there are no remaining participants 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 dialog that carries the <**createconference**> element is terminated. When this occurs, a MS 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.

Additional content of the <**createconference**> element specifies the parameters of the audio and/or video mixes.

An example of the creation of an audio conference is shown below. This conference 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>
            <asn ri="10s"/>
        </audiomix>
    </createconference>
</msml>
```

A.8.3 Dialog (dialog)

Definition

Dialogs are a class of objects that represent automated participants. Dialogs 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 the MS. Unlike network connections, dialogs are created and destroyed through MSML. The MS implements the dialog participant.

A Dialog is a generic reference to the set of resources, both media and control, that are used to create a simple or complex action. An atomic play or record is an example of a simple action.

The function that a Dialog instance fulfills is defined by a client and the language utilized. In this case, it is MOML.

MSML Dialog instances are instantiated through the <**dialogstart**> element.

Object Creation

All MSML objects except the Network Connection objects are created using MSML commands/elements.

The following example starts a MOML dialog on a connection.

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

A.8.4 Operator (oper)

Definition

An Operator is an object or class used to filter or transform a media stream. Operators have a media type and may be unidirectional or bidirectional.

Unidirectional operators reflect simple atomic functions, such as automatic gain control or filtering tones. Unidirectional operators have a single media input that is connected to the media stream from one object, and a single media output that 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.

The function that an Operator instance fulfills is defined by a client and the language utilized. In this case, it is MOML.

MSML Operator instances are instantiated when streams are created using a **<join>** element or modified using a **<modifystream>** element.

Object Creation

All MSML objects except Network Connection objects are created using MSML commands/elements.

Media Object Markup Language (MOML) Overview B

This chapter provides a brief overview of the Media Object Markup Language (MOML). The descriptions provide here are based on *Media Server Markup Language (MSML)*, Internet IETF Draft, saleem-msml-00, which was current during the development of the version of MSML Media Server Software described in this manual. Topics include:

B.1 Introduction

The Media Objects Markup Language (MOML) is a modular and extensible language to define media processing objects that execute on media servers. The base language defines a set of primitive media objects and provides tools to group primitives together and specify how they interact with each other. Clients can use the base MOML, or extend MOML, to create precisely tailored media processing objects that may be used as parts of application interactions with users or conferences or to transform media flowing internal to a media server. Interactive Voice Response (IVR) is an example of an application interaction with a user.

B.2 Primitives

MOML primitives perform a single function on a media stream such as audio generation, speech recognition, DTMF detection or gain adjustment.

MOML primitives can be divided into three main categories:

- Recognizers
- Transformers
- Sources and Sinks

B.2.1 Recognizers

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

The recognizers defined in the IETF draft include:

- **dtmf**

  DTMF input fulfills several roles. It is used to trigger events that affect media processing. It is also used to collect DTMF digits from a media stream, which are reported back to the application server.
- **faxdetect**

  Fax tone detection is used to detect the presence of the T.30 CNG tone in a media stream.
- **speech**

  Speech activates grammars or user input rules associated with speech recognition.
- **vad**

  Voice activity detection (VAD) is used to detect voice and silence when speech recognition is not required.

B.2.2 Transformers

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

The recognizers defined in the IETF draft include:

- **agc**

  Automatic Gain Control (AGC), which is the process where the media server automatically adjusts the gain of a media stream.
- **clamp**

  Used to filter DTMF tones from a media stream.
- **gain**

  Used to adjust the gain of a media stream by a specific amount.
- **gate**

  A simple filter that passes or halts media, regardless of the format of the media stream, based on the events it receives.
- **relay**

  A simple primitive that copies its input to its output.

B.2.3 Sources and Sinks

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

The sources and sinks defined in the IETF draft include:

- **play**

  Used to generate an audio stream.
- **record**

  Creates a recording.

- **dtmfgen**

  Originates one or more DTMF digits in sequence.
- **faxsend**

  Provides the functionality of a calling fax terminal.
- **faxrcv**

  Provides the functionality of a called fax terminal.

B.3 Reference to Examples

See Chapter 3, “Sample Use Case” for examples of the MSML control syntax that can be used to perform media control operations.

Glossary

3PCC: Third-Party Call Control

Application Server: An external server running applications that originate MSML requests to a Media Server (MS).

IETF: Internet Engineering Task Force

IVR: Interactive Voice Response

IP: Internet Protocol

Media Server: A general-purpose platform for executing real-time media processing tasks. It may be a single physical device or a logical function within a physical device.

MOML: Media Objects Markup Language

MSML: Media Sessions Markup Language

RTP: Real Time Protocol

SDP: Session Description Protocol

SIP: Session Initiation Protocol

XML: Extensible Markup Language