Video Messaging Server Interface Developer's Reference Manual

9000-62479-18

NMS COMMUNICATIONS

100 Crossing Boulevard
Framingham, MA 01702-5406 USA
www.nmscommunications.com

P/N 9000-62479-18

Revision history

Revision	Release date	Notes
1.0	March 2005	SRG, Video Access 1.0
1.1	September 2005	DEH, Video Access 2.0 Beta 1
1.2	October 2005	DEH, Video Access 2.0 Beta 2
1.3	December 2005	DEH, Video Access 2.0
1.4	February 2007	DEH, Video Access 3.0 Alpha
1.5	March 2007	AMO, Video Access 3.0 Beta 1
1.6	May 2007	PJP, Video Access 3.0 Beta 2
1.7	July 2007	PJP, Video Access 3.0
1.8	February 2009	DEH, Video Access 3.2
Last modified: January 22, 2009		

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

Table Of Contents

1 Introduction

The *Video Messaging Server Interface Developer's Reference Manual* targets video application developers who are using Natural Access. It defines telephony terms where applicable, but assumes that you are familiar with telephony concepts, switching, Natural Access, and the C programming language.

If you are not familiar with Natural Access, read the *Video Access Overview Manual* to learn about the Natural Access features that relate to Video Access before reviewing this manual.

2 Overview of the Video Messaging Server Interface

Video Messaging Server Interface overview

The Video Messaging Server Interface provides an application with a set of media processing capabilities for video and audio streams in an IPv4 or IPv6 network. These capabilities enable an application to play and record RTP audio and video streams, and to store data and retrieve data from 3GP file format files.

For information about the Video Messaging Server Interface components, architecture, and data flow, see the *Video Access Overview Manual*.

Video Access document set

The following table describes each of the manuals in the Video Access documentation set, along with guidelines for their use:

Manual	Description	Use this manual if...
Video Access Overview Manual	A general introduction to Video Access and its features.	You are new to Video Access. Start with this manual before proceeding to the *Video Mail Application Demonstration Manual.*
Video Mail Application Demonstration Program Manual	How to use *vmsamp*, a functional video mail application built on Video Access and supplied with the product.	You are new to Video Access and want to gain hands-on experience with Video Access technology and code before you start writing your own applications. The *vmsamp* application includes reference code for most of the data structures and API features described in the other Video Access manuals.
3G-324M Interface Developer's Reference Manual	How to use the 3G-324M Interface to connect with 3G-324M terminals capable of audio and video. This manual also describes the 3G-324M Interface capabilities and functions.	You are developing gateway functionality based on the 3G-324M Interface.

Manual	**Description**	**Use this manual if...**
Video Messaging Server Interface Developer's Reference Manual	How to play and record audio and video RTP media, and how to use the Video Messaging Server Interface.	Your application will use the Video Messaging Server Interface to process video and audio streams.
Video Access Utilities Manual	How to use the Video Access utilities that are available for manipulating 3GP files and monitoring 3G-324M calls.	You are responsible for Video Access content capture and analysis, or for the manipulation or troubleshooting of data generated or received by Video Access components. The utilities documented here can also be used to manipulate content created outside of Video Access.

Note: For an additional layer of detail about Video Access structures, refer to the Video Access header files.

3 Configuring the Video Messaging Server Interface

Overview of configuring the interface

The following topics discuss what you will need to know to configure the Video Messaging Server Interface. Specifically, the following topics are covered:

- Sample CG board configurations.
- Audio and video codec types that are supported.
- Setting up the Video Messaging Server Interface for various audio configurations:
    - Audio pass-through
    - Audio pass-through with silence detection
    - Audio transcoding
- How to set up the Video Messaging Server Interface to allow video server applications to forward video and audio streams.
- API calls used to setup multimedia pass-through on video channels.
- API calls used to setup multimedia pass-through on audio channels.
- Configuring endpoints to use RTCP audio/video synchronization.

Configuring the board

This topic provides sample files to use for configuring CG boards for a video server application with a 3G-324M gateway. Refer to the CG board installation and developer's manual for general information about configuring the board.

CG 6565 board sample configuration file

For the gateway portion:

- Twelve DSPs are reserved in the MUX_DEMUX pool for the 3G-324M interface, providing a total of 120 3G-324M MUX/DEMUX ports. Each DSP can support 10 MUX/DEMUX ports, available on timeslots 0 through 119 (specified in **ctaOpenServices**).

For the server portion:

- Eight DSPs are reserved in the IVR pool for the silence detection and ADI IVR operation, providing a total of 90 ports, available in timeslots 120 through 209 (specified in **ctaOpenServices**).
- Eighteen DSPs are reserved in the RTP pool for the audio transcoding (AMR and G.723), providing a total of 90 ports, available in timeslots 210 through 299 (specified in **ctaOpenServices**).

The video-specific settings in the following sample configuration file are shown in bold.

```
#################################################################
#
# video_mail_6565.cfg
#
# CG6565 Boot configuration file for Video Mail System
#
#################################################################
#################################################################
# CG6565 IP Address, subnet mask, and gateway IP address.
# Note: the IP configuration below is for a Ethernet Failover
# THIS CONFIGURATION FILE WILL FAIL UNLESS THE VARIABLE STRINGS
# BELOW ARE REPLACE WITH REAL IP ADDRESSES.
IPC.AddRoute[0].DestinationAddress = 10.118.7.133
IPC.AddRoute[0].Mask = 255.255.0.0
IPC.AddRoute[0].Interface = 1
#IPC.AddRoute[1].DestinationAddress = 0.0.0.0
#IPC.AddRoute[1].Mask = 0.0.0.0
#IPC.AddRoute[1].GatewayAddress = 10.1.0.1
#################################################################
IPv6.Link[0].Enable = YES
IPv6.Link[0].IPSec  = NO
IPv6.Link[0].MTU    = 1500
IPv6.Link[0].HopLimit = 64
IPv6.Link[0].EnablePing = YES
IPv6.Link[0].ICMPRateLimit = 100
IPv6.Link[0].NDAttempts = 3
IPv6.Link[0].NDRetranTimer = 1000
IPv6.Link[0].NDReachabilityTImer = 30000
IPv6.Link[1].Enable = YES
IPv6.Link[1].IPSec  = NO
IPv6.Link[1].MTU    = 1500
IPv6.Link[1].HopLimit = 128
IPv6.Link[1].EnablePing = YES
IPv6.Link[1].ICMPRateLimit = 100
IPv6.Link[1].NDAttempts = 3
IPv6.Link[1].NDRetranTimer = 1000
IPv6.Link[1].NDReachabilityTImer = 30000
```

```
#################################################################
# E1 SPECIFICS
TCPFiles                                    = nocc isd0
NetworkInterface.T1E1[0..15].Type           = E1
NetworkInterface.T1E1[0..15].Impedance      = G703_120_OHM
NetworkInterface.T1E1[0..15].LineCode       = HDB3
NetworkInterface.T1E1[0..15].FrameType      = CEPT
NetworkInterface.T1E1[0..15].SignalingType  = PRI
NetworkInterface.T1E1[0..15].D_Channel      = ISDN
DSPStream.VoiceIdleCode[0..15]              = 0xD5
DSPStream.SignalIdleCode[0..15]             = 0x09
Hdlc[0..3].Boot                             = YES
Hdlc[0..3].Hardware.TxTimeSlot              = 16
Hdlc[0..3].Hardware.RxTimeSlot              = 16
#################################################################
# CLOCK SETTINGS
Clocking.HBus.ClockMode                     = STANDALONE
Clocking.HBus.ClockSource                   = NETWORK
Clocking.HBus.ClockSourceNetwork            = 1
#################################################################
# DSP RELATED SETTINGS
#################################################################
DSP.C5x[0..95].Os                           = dspos6u
# DSP Libraries - E1
DSP.C5x[0..95].Libs                         = cg6kliba f_shared
# ----------------------------------------------------------
# Set up the voice processing DSP's in A_LAW (for E1)
# Set up the MUX DSP's in NO_LAW so they won't compand
# ----------------------------------------------------------
DSP.C5x[0..95].XLaw                         = A_LAW
DSP.C5x[0..11].XLaw                         = NO_LAW

# -----------------------------------------------------------------
# Very important for MUX DSP's in 3G-324M Interface configuration!
# -----------------------------------------------------------------
DSP.C5x[0..11].DataInQSize                  = 0x800
DSP.C5x[0..11].DspOutQStart                 = 0x2900
DSP.C5x[0..11].DspOutQSize                  = 0x900
#################################################################
# RESOURCE MANAGEMENT
#################################################################
#################################################################
# Resource Pool 1 - MUX
#################################################################
Resource[0].Name              = MUX_DEMUX
Resource[0].TCPs              = nocc isd0
Resource[0].DSPs              = 0 1 2 3 4 5 6 7 8 9 10 11
Resource[0].Size              = 120
Resource[0].StartTimeSlot     = 0
Resource[0].Definitions       = (mux.mux & mux.demux)
#################################################################
# Resource Pool 2 - IVR (for Silence Detection)
#################################################################
Resource[1].Name              = IVR
Resource[1].TCPs              = nocc isd0
Resource[1].DSPs              = 12 13 14 15 16 17 18 19
Resource[1].Size              = 90
Resource[1].StartTimeSlot     = 120
Resource[1].Definitions       = ( tone.gen  | dtmf.det_all | dtmf.dtmf_sil_clrdwn |\
(rvoice.rec_alaw & rvoice.play_alaw) )
#################################################################
# Resource Pool 3 - Fusion AMR/G723
#################################################################
#Resource[2].Name            = RTP
#Resource[2].TCPs            = nocc isd0
#Resource[2].DSPs            = 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
#Resource[2].Size            = 90
#Resource[2].StartTimeSlot   = 210
#Resource[2].Definitions     = ( (f_amr.cod & f_amr.dec) | \
#   (f_g723.cod & f_g723.dec) )
```

```
##############################################################################
# DOWNLOADABLE RUNTIME MODULES6krun
DLMFiles[0]                  = cg6565fusion
DLMFiles[1]                  = c6565igen
#########################################################
# DEBUG STUFF
DebugMask                   = 0x0
###############################################################
```

CG 6060 board sample configuration file

For the gateway portion:

- Six DSPs are reserved in the MUX_DEMUX pool for the 3G-324M interface, providing a total of 60 3G-324M MUX/DEMUX ports. Each DSP can support 10 MUX/DEMUX ports, available on timeslots 0 through 59 (specified in **ctaOpenServices**).

For the server portion:

- Four DSPs are reserved in the IVR pool for the silence detection and ADI IVR operation, providing a total of 40 ports, available in timeslots 60 through 99 (specified in **ctaOpenServices**).
- Eight DSPs are reserved in the RTP pool for the audio transcoding (AMR and G.723), providing a total of 40 ports, available in timeslots 100 through 139 (specified in **ctaOpenServices**).

The video-specific settings in the following sample configuration file are shown in bold.

```
###############################################################
#
# video_mail_6060.cfg
#
# CG6060 Boot configuration file for Video Mail System
#
###############################################################
###############################################################
# CG6060 IP Address, subnet mask, and gateway IP address.
# Note: the IP configuration below is for a Ethernet Failover
# THIS CONFIGURATION FILE WILL FAIL UNLESS THE VARIABLE STRINGS
# BELOW ARE REPLACE WITH REAL IP ADDRESSES.
IPC.AddRoute[0].DestinationAddress = 10.118.7.133
IPC.AddRoute[0].Mask = 255.255.0.0
IPC.AddRoute[0].Interface = 1
#IPC.AddRoute[1].DestinationAddress = 0.0.0.0
#IPC.AddRoute[1].Mask = 0.0.0.0
#IPC.AddRoute[1].GatewayAddress = 10.1.0.1
###############################################################
IPv6.Link[0].Enable = YES
IPv6.Link[0].IPSec  = NO
IPv6.Link[0].MTU    = 1500
IPv6.Link[0].HopLimit = 64
IPv6.Link[0].EnablePing = YES
IPv6.Link[0].ICMPRateLimit = 100
IPv6.Link[0].NDAttempts = 3
IPv6.Link[0].NDRetranTimer = 1000
IPv6.Link[0].NDReachabilityTImer = 30000
IPv6.Link[1].Enable = YES
IPv6.Link[1].IPSec  = NO
IPv6.Link[1].MTU    = 1500
IPv6.Link[1].HopLimit = 128
IPv6.Link[1].EnablePing = YES
IPv6.Link[1].ICMPRateLimit = 100
IPv6.Link[1].NDAttempts = 3
IPv6.Link[1].NDRetranTimer = 1000
IPv6.Link[1].NDReachabilityTImer = 30000
```

```
#################################################################
# E1 SPECIFICS
TCPFiles                                   = nocc isd0
NetworkInterface.T1E1[0..15].Type          = E1
NetworkInterface.T1E1[0..15].Impedance     = G703_120_OHM
NetworkInterface.T1E1[0..15].LineCode      = HDB3
NetworkInterface.T1E1[0..15].FrameType     = CEPT
NetworkInterface.T1E1[0..15].SignalingType = PRI
NetworkInterface.T1E1[0..15].D_Channel     = ISDN
DSPStream.VoiceIdleCode[0..15]             = 0xD5
DSPStream.SignalIdleCode[0..15]            = 0x09
Hdlc[0..3].Boot                            = YES
Hdlc[0..3].Hardware.TxTimeSlot             = 16
#################################################################
# CLOCK SETTINGS
Clocking.HBus.ClockMode                    = STANDALONE
Clocking.HBus.ClockSource                  = NETWORK
Clocking.HBus.ClockSourceNetwork           = 1
#################################################################
# DSP RELATED SETTINGS
#################################################################
DSP.C5x[0..47].Os                          = dspos6u
# DSP Libraries - E1
DSP.C5x[0..47].Libs                        = cg6kliba f_shared
# -----------------------------------------------------------
# Set up the voice processing DSP's in A_LAW (for E1)
# Set up the MUX DSP's in NO_LAW so they won't compand
# -----------------------------------------------------------
DSP.C5x[0..47].XLaw                        = A_LAW
DSP.C5x[0..5].XLaw                        = NO_LAW

# ------------------------------------------------------------------
# Very important for MUX DSP's in 3G-324M Interface configuration!
# ------------------------------------------------------------------
DSP.C5x[0..5].DataInQSize                 = 0x800
DSP.C5x[0..5].DspOutQStart                = 0x2900
DSP.C5x[0..5].DspOutQSize                 = 0x900
#################################################################
# RESOURCE MANAGEMENT
#################################################################
#################################################################
# Resource Pool 1 - MUX
#################################################################
Resource[0].Name              = MUX_DEMUX
Resource[0].TCPs              = nocc
Resource[0].DSPs              = 0 1 2 3 4 5
Resource[0].Size              = 60
Resource[0].StartTimeSlot     = 0
Resource[0].Definitions       = (mux.mux & mux.demux)
#################################################################
# Resource Pool 2 - IVR (for Silence Detection)
#################################################################
Resource[1].Name              = IVR
Resource[1].TCPs              = nocc
Resource[1].DSPs              = 6 7 8 9
Resource[1].Size              = 40
Resource[1].StartTimeSlot     = 60
Resource[1].Definitions       = ( tone.gen  | dtmf.det_all | dtmf.dtmf_sil_clrdwn |\
(rvoice.rec_alaw & rvoice.play_alaw) )
#################################################################
# Resource Pool 3 - Fusion AMR/G723
#################################################################
#Resource[2].Name             = RTP
#Resource[2].TCPs             = nocc
#Resource[2].DSPs             = 10 11 12 13 14 15 16 17
#Resource[2].Size             = 40
#Resource[2].StartTimeSlot    = 100
#Resource[2].Definitions      = ( (f_amr.cod & f_amr.dec) | \
#   (f_g723.cod & f_g723.dec) )
```

```
###############################################################################
# DOWNLOADABLE RUNTIME MODULES6krun
DLMFiles[0]                  = cg6060fusion
DLMFiles[1]                  = c6060igen
###########################################################
# DEBUG STUFF
DebugMask                   = 0x0
#################################################################
```

CG 6000 board sample configuration file

For the gateway portion:

- Twelve DSPs are reserved in the MUX_DEMUX pool for the 3G-324M interface, providing a total of 48 3G-324M MUX/DEMUX ports. Each DSP can support four MUX/DEMUX ports, available on timeslots 0 through 47 (specified in **ctaOpenServices**).

For the server portion:

- Eight DSPs are reserved in the IVR pool for the silence detection and ADI IVR operation, providing a total of 32 ports, available in timeslots 48 through 79 (specified in **ctaOpenServices**).
- Eight DSPs are reserved in the RTP pool for the audio transcoding (AMR and G.723), providing a total of 32 ports, available in timeslots 80 through 111 (specified in **ctaOpenServices**).

The video-specific settings in the following sample configuration file are shown in bold.

```
#################################################################
#
# video_mail_6000.cfg
#
# CG6000 Boot configuration file for Video Mail System
#
#
#################################################################
#################################################################
# CG6000 IP Address, subnet mask, and gateway IP address.
# Note: the IP configuration below is for a Ethernet Failover
# THIS CONFIGURATION FILE WILL FAIL UNLESS THE VARIABLE STRINGS
# BELOW ARE REPLACE WITH REAL IP ADDRESSES.
IPC.AddRoute[0].DestinationAddress         = 10.118.7.121
IPC.AddRoute[0].Mask                       = 255.255.0.0
IPC.AddRoute[0].Interface                  = 1
#IPC.AddRoute[1].DestinationAddress        = 0.0.0.0
#IPC.AddRoute[1].Mask                      = 0.0.0.0
#IPC.AddRoute[1].GatewayAddress            = 10.1.0.1
#################################################################
IPv6.Link[0].Enable = YES
IPv6.Link[0].IPSec  = NO
IPv6.Link[0].MTU    = 1500
IPv6.Link[0].HopLimit = 64
IPv6.Link[0].EnablePing = YES
IPv6.Link[0].ICMPRateLimit = 100
IPv6.Link[0].NDAttempts = 3
IPv6.Link[0].NDRetranTimer = 1000
IPv6.Link[0].NDReachabilityTImer = 30000
IPv6.Link[1].Enable = YES
IPv6.Link[1].IPSec  = NO
IPv6.Link[1].MTU    = 1500
IPv6.Link[1].HopLimit = 128
IPv6.Link[1].EnablePing = YES
IPv6.Link[1].ICMPRateLimit = 100
IPv6.Link[1].NDAttempts = 3
```

```
IPv6.Link[1].NDRetranTimer = 1000
IPv6.Link[1].NDReachabilityTImer = 30000
################################################################
# E1 SPECIFICS
TCPFiles                                   = nocc isd0
NetworkInterface.T1E1[0..3].Type           = E1
NetworkInterface.T1E1[0..3].Impedance      = G703_120_OHM
NetworkInterface.T1E1[0..3].LineCode       = HDB3
NetworkInterface.T1E1[0..3].FrameType      = CEPT
NetworkInterface.T1E1[0..3].SignalingType  = PRI
NetworkInterface.T1E1[0..3].D_Channel      = ISDN
DSPStream.VoiceIdleCode[0..3]              = 0xD5
DSPStream.SignalIdleCode[0..3]             = 0xB
Hdlc[0,3,6,9].Boot                         = YES
Hdlc[0,3,6,9].Comet.TxTimeSlot             = 16
Hdlc[0,3,6,9].Comet.RxTimeSlot             = 16
################################################################
MaxChannels = 150
################################################################
# CLOCK SETTINGS
Clocking.HBus.ClockMode                    = STANDALONE
Clocking.HBus.ClockSource                  = NETWORK
Clocking.HBus.ClockSourceNetwork           = 1
################################################################
################################################################
# DSP RELATED SETTINGS
################################################################
DSP.C5x[0].Files                           = qtsignal callp tone ptf dtmf echo mf
DSP.C5x[1..31].DataReqTimeOffset           = 7
# DSP Libraries - E1
DSP.C5x[0..31].Libs                        = cg6kliba f_shared
# ---------------------------------------------------------
# Set up the voice processing DSP's in A_LAW (for E1)
# Set up the MUX DSP's in NO_LAW so they won't compand
# ---------------------------------------------------------
DSP.C5x[0..31].XLaw                        = A_LAW
DSP.C5x[1..12].XLaw                        = NO_LAW
# ----------------------------------------------------------------
# Very important for MUX DSP's in 3G-324M Interface configuration!
# ----------------------------------------------------------------
DSP.C5x[1..12].DataInQSize                 = 0x2D0
DSP.C5x[1..12].DspOutQStart                = 0xFB50
DSP.C5x[1..12].DspOutQSize                 = 0x3A0
################################################################
################################################################
# RESOURCE MANAGEMENT
#
################################################################
################################################################
# Resource Pool 1 - MUX
################################################################
Resource[0].Name              = MUX_DEMUX
Resource[0].TCPs              = nocc isd0
Resource[0].DSPs              = 1 2 3 4 5 6 7 8 9 10 11 12
Resource[0].Size              = 48
Resource[0].StartTimeSlot     = 0
Resource[0].Definitions       = (mux.mux & mux.demux)
################################################################
# Resource Pool 2 - IVR (for Silence Detection)
################################################################
Resource[1].Name              = IVR
Resource[1].TCPs              = nocc isd0
Resource[1].DSPs              = 13 14 15 16 17 18 19 20
Resource[1].Size              = 32
Resource[1].StartTimeSlot     = 48
Resource[1].Definitions       = ( tone.gen  | dtmf.det_all | dtmf.dtmf_sil_clrdwn |\
(rvoice.rec_alaw & rvoice.play_alaw) )
```

```
#################################################################
# Resource Pool 3 - Fusion AMR/G723
#################################################################
Resource[2].Name              = RTP
Resource[2].TCPs              = nocc isd0
Resource[2].DSPs              = 21 22 23 24 25 26 27 28
Resource[2].Size              = 32
Resource[2].StartTimeSlot     = 80
Resource[2].Definitions       = ( (f_amr.cod & f_amr.dec) | \
(f_g723.cod & f_g723.dec) )
#################################################################
# DOWNLOADABLE RUNTIME MODULES
DLMFiles[0]                   = cg6krun
DLMFiles[1]                   = cg6kfusion
DLMFiles[2]                   = isdngen
#################################################################
#################################################################
# DEBUG STUFF
DebugMask                     = 0x00
###############################################################
```

Audio and video formats

A single audio and video codec can be selected and enabled for the duration of a video call. The following table presents the audio and video codecs that are supported on pass-through configurations (no audio transcoding):

Type	Media codec
Host audio	• AMR-NB in NMS packetized format. • G.723-1 in NMS packetized format (cannot be used with 3GP files). • G.711 in NMS packetized format (cannot be used with 3GP files).
Host video	• H.263 baseline level 10 to 30 in NMS packetized format. • H.263+ profile 3 level 10 to 30 in NMS packetized format. • H.264 baseline profile level 1 – 1.2 in NMS packetized format. • ISO/IEC 14496-2 MPEG-4 simple profile level 0 to 3 in NMS packetized format.
IP audio	• AMR narrow band audio, 3GPP version 5.3, RFC 3267 compliant for RTP payload formats. • G.723.1 • G.711 The following restrictions apply for conformance with RFC 3267: • Received codec mode requests (CMR) are not supported. • Only octet-aligned mode is supported. • Forward error correction, interleaving, robust sorting, UEP/UED bit error detection schemes, and multi-channel payloads are not supported.
IP video	• H.263 baseline video, as specified in annex X (level 10 to 30), RFC 2190 and RFC 2429 compliant for RTP packetization. • H.263+ profile 3, as specified in annex X (level 10 to 30), RFC 2429 compliant for RTP packetization. • H.264 baseline profile 1 – 1.2 video, RFC 3984 compliant (no interleaving) for RTP payload. • MPEG-4 simple profile level 0 to 3 video, RFC 3016 compliant for RTP payload.

The application can set up the Video Messaging Server Interface to perform audio transcoding, at the expense of port density. The audio transcoding configuration supports any NMS-standard ADI/IVR or Fusion audio codec. However, only AMR can be used as the IVR codec for 3GP file storage. For information, see *Defining an audio transcoding configuration* on page 27.

See the *ADI Service Developer's Reference Manual* for the list of supported host codecs and file formats. See the *Fusion Developer's Manual* for a list of supported codecs on an IP interface.

Defining the Video Messaging Server configuration

The ADI service and the MSPP service provide the flexibility to satisfy the following requirements:

- Selected audio codec types for the network and for storage
- Optimized for audio quality and board resources by allowing applications to audio transcode only when necessary
- Audio silence detection

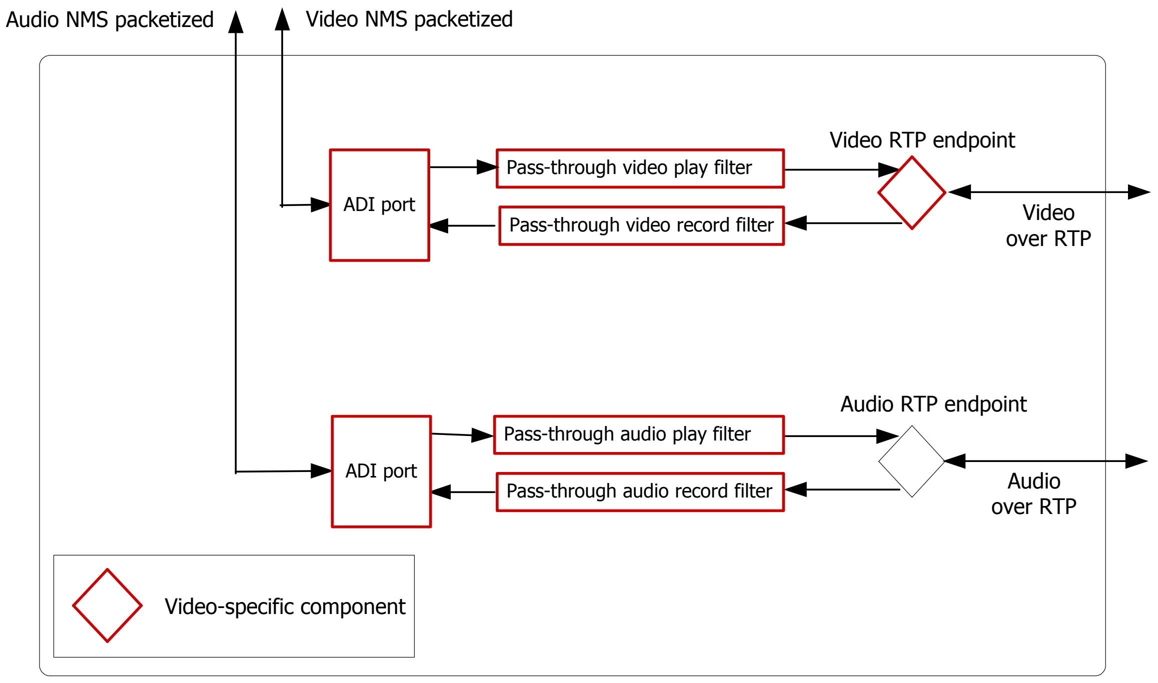
Note: Video path is always configured in pass-through mode because the Video Messaging Server Interface does not provide video transcoding capability. This configuration requires no DSP resources on the CG board, and is the same for all audio configurations.

Defining an audio pass-through configuration

In the audio pass-through configuration, the media payload of the audio and video RTP packets corresponds exactly to what the application receives from or submits to Video Access. Pass-through means that no media transformation is applied, and therefore this configuration does not consume any DSP resources on the board.

Voice quality is not degraded because no audio transcoding is performed. The recorded data can be played back only to terminals that support this media encoding.

The following illustration shows an audio pass-through configuration:



When no audio transcoding or silence detection is required, the application creates the following endpoints:

- A video endpoint (to handle MPEG-4, H.263, or H.264)
- An MSPP audio endpoint (for use with AMR, G.723, or G.711)

The application associates each endpoint with a corresponding ADI port and calls **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, or **adiRecordMMAsync** to create the filters and initiate pass-through play or record. Refer to *Video enhancements to the ADI service* on page 51 and *Video enhancements to the MSPP service* on page 69 for more information.

Note: The application can synchronize audio and video streams on video I-frame detection. It can also stop the recording of audio and video streams simultaneously. See **adiRecordMMToMemory** or **adiRecordMMAsync** for more information.

Function sequence

The following tables show a typical function sequence for an audio-pass through configuration. This sequence shows:

- Initialization
- Play operation
- Record operation

The sequence contains both video-specific and non-video-specific function calls.

Initialization

Step	Call	Description
1	**ctaCreateQueue** (&queuehd)	Creates a CTA queue.
2	**ctaCreateContext** (&video_ctahd, queuehd)	Creates a CTA context for the video channel.
3	**ctaCreateContext** (&audio_ctahd, queuehd)	Creates a CTA context for the audio channel.
4	**ctaOpenServices** (video_ctahd, svclist, nsvcs)	Creates an ADI port for the pass-through video. The pass-through video channel consumes no DSP resources on the CG board. Therefore, the svclist parameter of **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to 0 to prevent allocation of DSP resources for this channel on the CG board. • The svclist.mvipaddr.stream and svclist.mvipaddr.timeslot parameters are not applicable when the mvipaddr.mode parameter is set to 0. Set these parameters to 0 for the pass-through video channel. The ADI service and the MSPP service are in the list of services.
5	**ctaOpenServices** (audio_ctahd, svclist, nsvcs)	Creates an ADI port for the pass-through audio. The pass-through audio channel consumes no DSP resources on the CG board. Therefore, the svclist parameter of **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to 0. • The svclist.mvipaddr.stream and svclist.mvipaddr.timeslot parameters are not applicable when the mvipaddr.mode parameter is set to 0. Set these parameters to 0. The ADI service and the MSPP service are in the list of services.
6	**mspCreateEndpoint** (video_ctahd, &video_ephd)	Creates a video RTP endpoint.
7	**mspCreateEndpoint** (audio_ctahd, &audio_ephd)	Creates an audio RTP endpoint.

Step	Call	Description
8	**mspGetFilterHandle** (video_ctahd, video_ephd, &cg_video_ephd)	Translates the video MSPP ***ephd*** to a CG board ***ephd***.
9	**mspGetFilterHandle** (audio_ctahd, audio_ephd, &cg_audio_ephd)	Translates the audio MSPP ***ephd*** to a CG board ***ephd***.

Play operation

Step	Call	Description
1	**adiPlayMMFromMemory** (video_ctahd, cg_video_ephd,,,,) or **adiPlayMMAsync** (video_ctahd, cg_video_ephd,,,,)	Begins playing video portion of message.
2	**adiPlayMMFromMemory** (audio_ctahd, cg_audio_ephd,,,,) or **adiPlayMMAsync** (audio_ctahd, cg_audio_ephd,,,,)	Begins playing audio portion of message.
3	**adiStopPlaying** (video_ctahd)	Stops playing video portion of message.
4	**adiStopPlaying** (audio_ctahd)	Stops playing audio portion of message.

Record operation

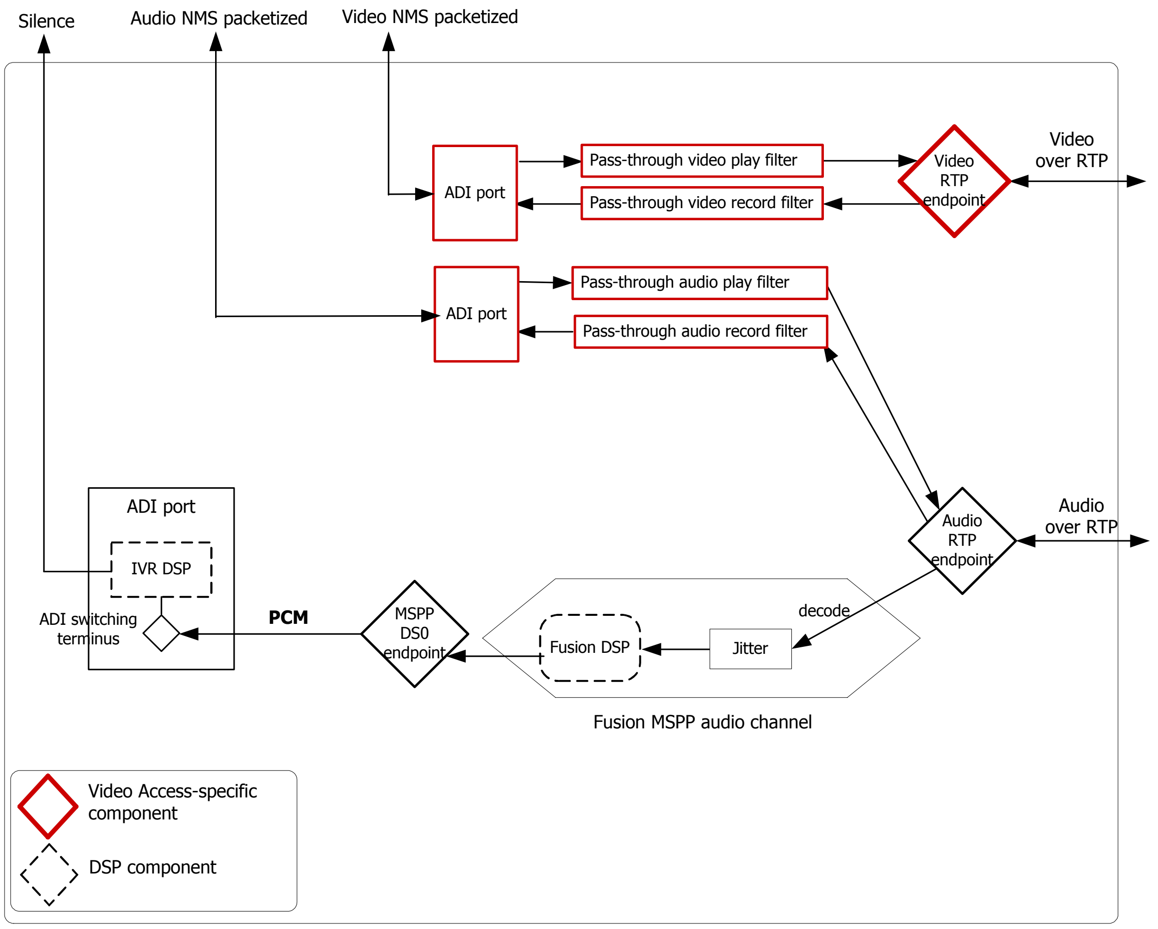
Step	Call	Description
1	**adiRecordMMToMemory** (video_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsync** (video_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording video portion of message. The application passes both the audio board handle and the video board handle to the video record function, reflecting the inter-dependence between the audio and video streams (for example, for I-frame detection).
2	**adiRecordMMToMemory** (audio_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsync** (audio_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording audio portion of message. The application passes both the audio board handle and the video board handle to the audio record function, reflecting the inter-dependence between the audio and video streams (for example, for I-frame detection).
3	**adiStopRecording** (video_ctahd)	Stops recording video portion of message.
4	**adiStopRecording** (audio_ctahd)	Stops recording audio portion of message.

Defining an audio pass-through with silence detection configuration

The audio pass-through with silence detection configuration is similar to the audio pass-through configuration for the media paths in that no media transformation is applied and no DSP resources are used for media transformation.

The difference is that a silence detector is inserted, for example, to avoid recording the initial silence of a voice message. This silence detection requires a DSP port and an MSPP connection between the DSP port and the audio RTP endpoint.

The following illustration shows an audio pass-through with silence detection configuration:



When silence detection is required for controlling the record function, the application must:

- Set up the audio pass-though path.
- Connect the audio RTP endpoint to an MSPP channel that is attached to an IVR silence detection resource. For more information, see the *ADI Service Developer's Reference Manual* and the *Fusion Developer's Manual*.
- Associate each endpoint to an ADI port.
- Call **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, or **adiRecordMMAsync** to create the filters and initiate pass-through play or record.

See *Video enhancements to the ADI service* on page 51 and *Video enhancements to the MSPP service* on page 69 for more information.

Function sequence

The following tables present a typical function sequence for an audio-pass through with silence detection configuration. This sequence shows

- Initialization
- Play operation
- Record operation

This function sequence contains both video-specific and non-video-specific calls.

Initialization

Step	Call	Description
1	**ctaCreateQueue** (&queuehd)	Creates a CTA queue.
2	**ctaCreateContext** (&video_ctahd, queuehd)	Creates a CTA context for the video channel.
3	**ctaCreateContext** (&audio_ctahd, queuehd)	Creates a CTA context for the audio channel. A single context can be used for the ADI port and the Fusion channel.
4	**ctaOpenServices** (video_ctahd, svclist, nsvcs)	Creates an ADI port for the pass-through video. The pass-through video channel consumes no DSP resources on the CG board. Therefore, the svclist parameter of **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to 0 to prevent allocation of DSP resources for this channel on the CG board. • The svclist.mvipaddr.stream and svclist.mvipaddr.timeslot parameters are not applicable when the mvipaddr.mode parameter is set to 0. Set these parameters to 0 for the pass-through video channel. The ADI service and the MSPP service are in the list of services.

Step	Call	Description
5	**ctaOpenServices** (audio_ctahd, svclist, nsvcs)	Creates an ADI port and opens the MSPP service for audio. The audio channel requires an ADI DSP port for silence detection. Therefore, the svclist parameter for the ADI service in **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to ADI_VOICE_DUPLEX to allocate DSP resources for this channel on the CG board. • Set the svclist.mvipaddr.stream parameter to 16 or 0. NMS recommends 0. • Set the svclist.mvipaddr.timeslot parameter to the desired DSP port. Valid range is 0 - 127. See the *ADI Service Developers Reference Manual*. To open the MSPP service, and support a DS0 endpoint in the Fusion channel, the svclist parameter for the MSPP service in **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to ADI_VOICE_DUPLEX to allocate DSP resources for this channel on the CG board. • Set the svclist.mvipaddr.stream parameter to 16 or 0. NMS recommends 0. • Set the svclist.mvipaddr.timeslot parameter to the desired DSP port. Valid range is 0 - 127. See the *MSPP Service Developers Reference Manual*. Both the ADI service and the MSPP service must be in the list of services.
6	**nccStartProtocol** (audio_ctahd, nocc)	Starts the NOCC protocol on the audio channel to enable silence detection, transcoding, and play/record operation on the audio channel.
7	**mspCreateEndpoint** (video_ctahd, &video_ephd)	Creates a video RTP endpoint.
8	**mspCreateEndpoint** (audio_ctahd, &audio_ephd)	Creates an audio RTP endpoint.
9	**mspGetFilterHandle** (video_ctahd, video_ephd, &cg_video_ephd)	Translates the video MSPP ***ephd*** to a CG board ***ephd***.
10	**mspGetFilterHandle** (audio_ctahd, audio_ephd, &cg_audio_ephd)	Translates the audio MSPP ***ephd*** to a CG board ***ephd***.
11	**mspCreateEndpoint** (audio_ctahd, &sd_ephd)	Creates a DS0 endpoint for the silence detection path.
12	**mspCreateChannel** (audio_ctahd, &sd_chhd)	Creates an AMR MSPP channel for the silence detection path.

Step	Call	Description
13	**mspConnect** (audio_ephd, sd_chhd, sd_ephd)	Connects the audio RTP endpoint, the AMR channel, and the DS0 endpoint together for the silence detection path.
14	**swiMakeConnection** (&inp, &out)	Connects the DS0 endpoint to the audio DSP port declared in svclist.mvipaddr of **ctaOpenServices**. **Note:** The connection can be simplex.

Play operation

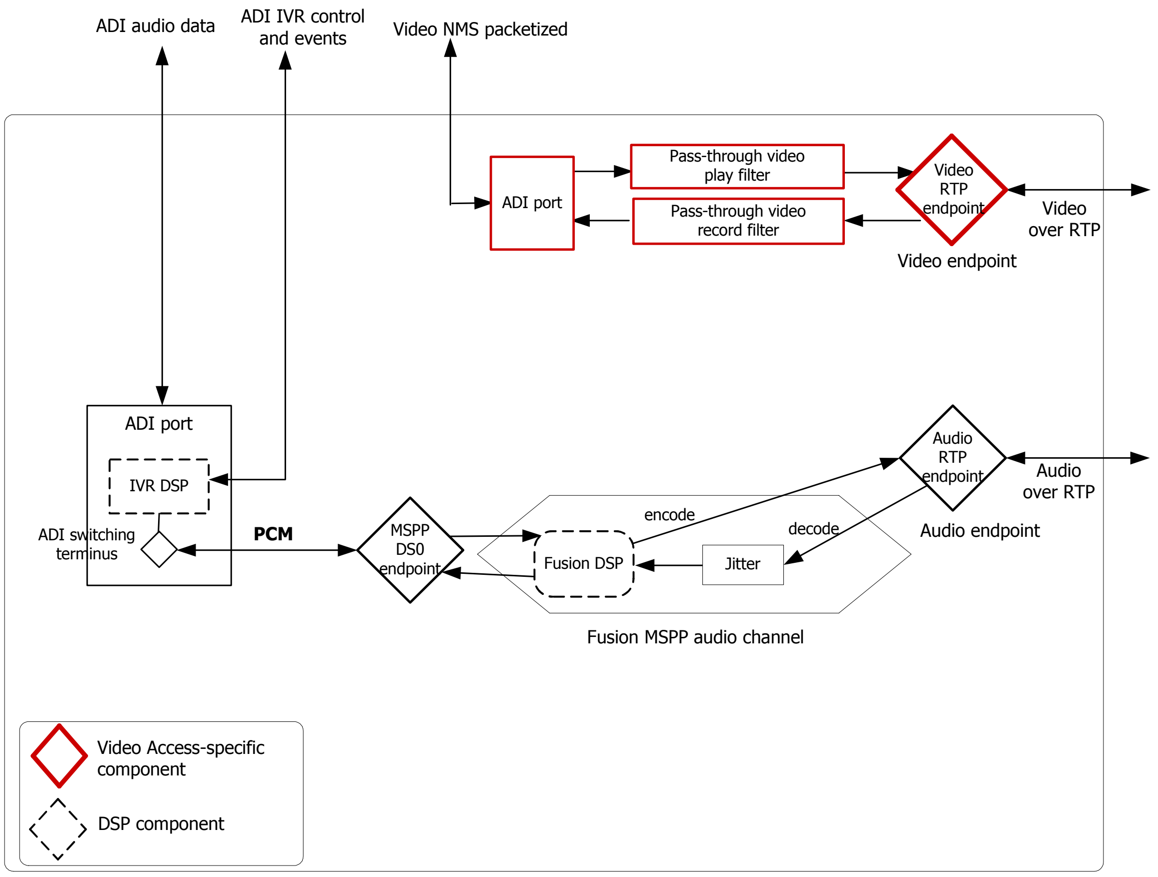
Step	Call	Description
1	**adiPlayMMFromMemory** (pvideo_ctahd, cg_video_ephd,,,,) or **adiPlayMMAsync** (pvideo_ctahd, cg_video_ephd,,,,)	Begins playing video portion of message.
2	**adiPlayMMFromMemory** (paudio_ctahd, cg_audio_ephd,,,,) or **adiPlayMMAsync** (pvideo_ctahd, cg_audio_ephd,,,,)	Begins playing of audio portion of message.
3	**adiStopPlaying** (pvideo_ctahd)	Stops playing video portion of message.
4	**adiStopPlaying** (paudio_ctahd)	Stops playing audio portion of message.

Record operation

Step	Call	Description
1	**mspEnableChannel** (sd_chhd)	Enables the AMR MSPP channel so that the incoming flow can reach the silence detector.
2	**adiRecordMMToMemory** (video_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsync** (video_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording video portion of message. The application passes both the audio board handle and the video board handle to the video record function, reflecting the inter-dependence between the audio and video streams (for example, for I-frame detection).
3	**adiRecordMMToMemory** (audio_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsync** (audio_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording of audio portion of message. Initial silence can be skipped and audio record can be aborted if a long silence is detected during the record operation. The application passes both the audio board handle and the video board handle to the audio record function, allowing the inter-dependence between the audio and video streams (for example, for I-frame detection).
4	**adiStopRecording** (video_ctahd)	Stops recording video portion of message.
5	**adiStopRecording** (audio_ctahd)	Stops recording audio portion of message.
6	**mspDisableChannel** (sd_chhd)	Disables the AMR MSPP channel.

Defining an audio transcoding configuration

In the audio transcoding configuration, instead of playing and recording audio data as is (pass-through configuration), the application can use another audio codec on the IP side or another media format on the host. The following illustration shows an audio transcoding configuration:



In this configuration, the application creates a standard Fusion path and switches the PCM audio stream to a standard ADI play/record resource (a DSP resource) allocated to support ADI service operations. For more information, see the *ADI Service Developer's Reference Manual* and the *Fusion Developer's Manual*.

The application uses the same ADI multimedia functions to perform play and record as in the pass-through configuration. By using these functions, the application can enable audio silence detection and synchronize the audio and video streams on video I-frame detection. It can also stop the recording of audio and video streams simultaneously.

Audio streams recorded in the audio transcoding configuration cannot be played back in a pass-through configuration, and vice-versa, without a conversion between NMS-packetized format and a raw audio bit stream. The one exception is when using the NMS Multimedia File Interface library functions for playing and recording 3GP files. In this case, the application can use these functions to convert AMR audio streams stored in or retrieved from 3GP format to NMS-packetized format or raw format. Once converted, the data can then be used in the pass-through configuration or the audio transcoding configuration, as shown in the following table:

Data format	**Configuration**
NMS-packetized	Pass-through
Raw	Audio transcoding

For more information on the Multimedia File Interface library functions, see *Multimedia File Interface Library overview* on page 81.

With an audio transcoding configuration, audio data is transcoded from and to whatever IP format is desired in a regular Fusion channel. The data can be recorded and played back in any format supported by the ADI service, but must be in AMR format, if using 3GP files. As in all configurations, video data is passed-through.

Function sequence

The following tables present a typical function sequence for an audio transcoding configuration. This sequence shows

- Initialization
- Play operation
- Record operation

This function sequence contains both video-specific and non-video-specific calls.

Initialization

Step	**Call**	**Description**
1	**ctaCreateQueue** (&queuehd)	Creates a CTA queue.
2	**ctaCreateContext** (&video_ctahd, queuehd)	Creates a CTA context for the video channel.
3	**ctaCreateContext** (&audio_ctahd, queuehd)	Creates a CTA context for the audio ADI port and the Fusion channel. A single context can be used for the audio ADI port and the Fusion channel or two contexts can be created.

Step	Call	Description
4	**ctaOpenServices** (video_ctahd, svclist, nsvcs)	Creates an ADI port for the pass-through video. The pass-through video channel consumes no DSP resources on the CG board. Therefore, the svclist parameter of **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to 0 to prevent allocation of DSP resources for this channel on the CG board. • The svclist.mvipaddr.stream and svclist.mvipaddr.timeslot parameters are not applicable when the mvipaddr.mode parameter is set to 0. Set these parameters to 0 for the pass-through video channel. The ADI service and the MSPP service are in the list of services.
5	**ctaOpenServices** (audio_ctahd, svclist, nsvcs)	Creates an ADI port and opens the MSPP service for audio. The audio channel requires an ADI DSP port for silence detection. Therefore, the svclist parameter for the ADI service in **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to ADI_VOICE_DUPLEX to allocate DSP resources for this channel on the CG board. • Set the svclist.mvipaddr.stream parameter to 16 or 0. (NMS recommends 0). • Set the svclist.mvipaddr.timeslot parameter to the desired DSP port. Valid range is 0 - 127. See the *ADI Service Developers Reference Manual*. To open the MSPP service, and support a DS0 endpoint in the Fusion channel, the svclist parameter for the MSPP service in **ctaOpenServices** must contain the following settings: • Set the svclist.mvipaddr.mode parameter to ADI_VOICE_DUPLEX to allocate DSP resources for this channel on the CG board. • Set the svclist.mvipaddr.stream parameter to 16 or 0. (NMS recommends 0.) • Set the svclist.mvipaddr.timeslot parameter to the desired DSP port. Valid range is 0 - 127. See the *MSPP Service Developers Reference Manual*. Both the ADI service and the MSPP service must be in the list of services.
6	**nccStartProtocol** (audio_ctahd, nocc)	Starts the NOCC protocol on the audio ADI port channel to enable silence detection, transcoding, and play/record operation on the audio channel.
7	**mspCreateEndpoint** (video_ctahd, &video_ephd)	Creates a video RTP endpoint.
8	**mspCreateEndpoint** (audio_ctahd, &audio_ephd)	Creates an audio MSPP RTP endpoint.

Step	Call	Description
9	**mspGetFilterHandle** (video_ctahd, video_ephd, &cg_video_ephd)	Translates the video MSPP ***ephd*** to a CG board ***ephd***. **Note:** In this configuration, only the video_ephd is translated.
10	**mspCreateEndpoint** (audio_ctahd, &txc_ephd)	Creates an MSPP DS0 endpoint for the audio transcoding path. The timeslot number provided must equal the one specified in **ctaOpenServices** for the MSPP service.
11	**mspCreateChannel** (audio_ctahd, &txc_chhd)	Creates an AMR MSPP channel for the transcoding path.
12	**mspConnect** (audio_ephd, txc_chhd, txc_ephd)	Connects the audio RTP endpoint, the AMR channel, and the MSPP DS0 endpoint together.
13	**swiMakeConnection** (&inp, &out)	Connects the MSPP DS0 endpoint and the ADI DSP port. Timeslot numbers are those specified in respective svclist.mvipaddr of **ctaOpenServices**. **Note:** The connection must be duplex to support play and record.

Play operation

Step	Call	Description
1	**adiPlayMMFromMemory** (video_ctahd, cg_video_ephd,,,,) or **adiPlayMMAsync** (video_ctahd, cg_video_ephd,,,,)	Begins playing video portion of message.
2	**adiPlayMMFromMemory** (audio_ctahd, cg_audio_ephd,,,,) or **adiPlayMMAsync** (audio_ctahd, cg_audio_ephd,,,,)	Begins playing audio portion of message.
3	**adiStopPlaying** (video_ctahd)	Stops playing video portion of message.
4	**adiStopPlaying** (audio_ctahd)	Stops playing audio portion of message.

Record operation

Step	Call	Description
1	**adiRecordMMToMemory** (video_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsync** (video_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording video portion of message. The application passes both the audio board handle and the video board handle to the video record function, allowing the inter-dependence between the audio and video streams (for example, for I-frame detection).
2	**adiRecordMMToMemory** (audio_ctahd, cg_video_ephd, cg_audio_ephd) or **adiRecordMMAsnyc** (audio_ctahd, cg_video_ephd, cg_audio_ephd)	Begins recording audio portion of message. Initial silence can be skipped and audio record can be aborted if a long silence is detected during the record operation. The application passes both the audio board handle and the video board handle to the audio record function, allowing the inter-dependence between the audio and video streams (for example, for I-frame detection).
3	**adiStopRecording** (video_ctahd)	Stops recording video portion of message.
4	**adiStopRecording** (audio_ctahd)	Stops recording audio portion of message.

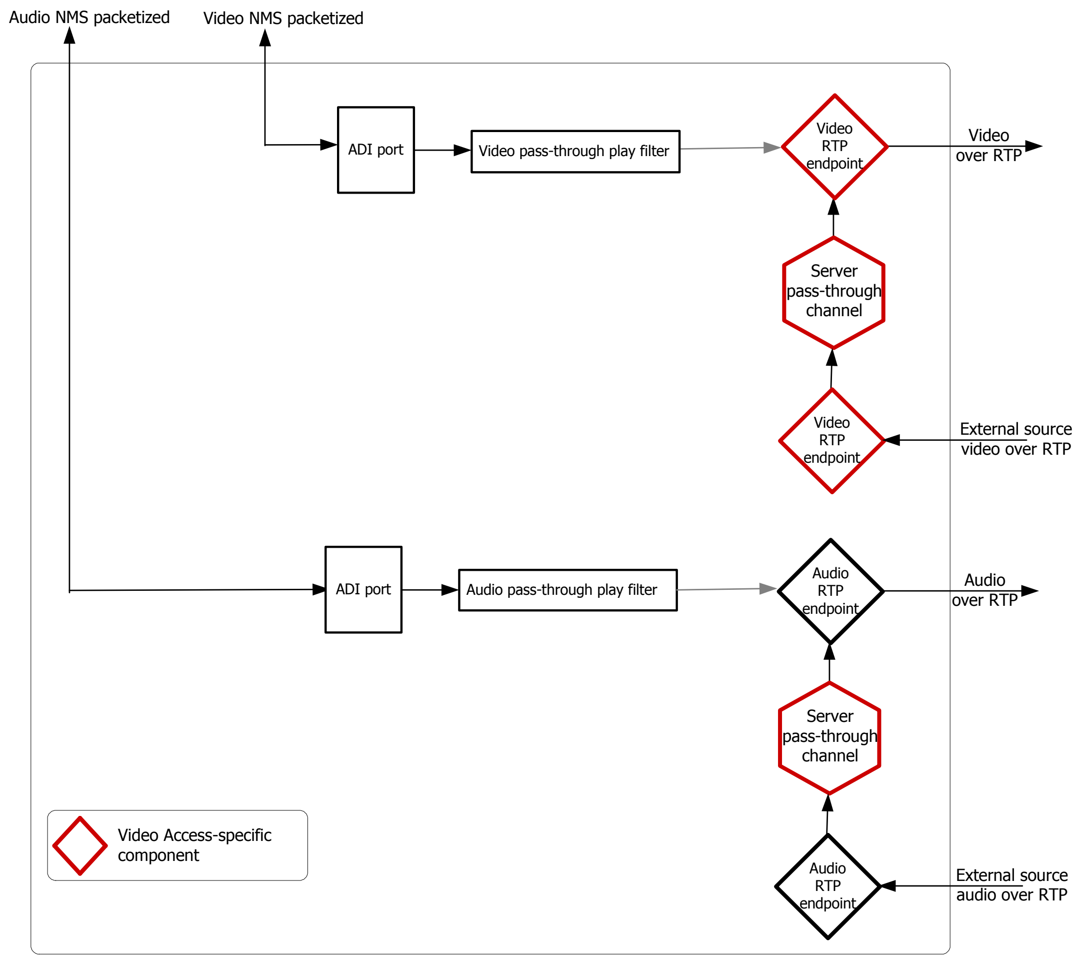
Defining a media bridging configuration

A media bridging configuration allows video server applications to forward video and audio streams:

- From external IP sources to 3G-324M terminals.
- From 3G-324M terminals to external IP devices.
- In both directions at once.

This configuration is typically a temporary modification of one of the previous configurations. For example, after having played an audio/video file, an application can switch to an external incoming RTP stream for a while, and later on, switch back to the host to play another file.

The following illustration shows a media bridging configuration derived from a playback configuration:


Audio NMS packetized
Video NMS packetized
ADI port
Video pass-through play filter
Video RTP endpoint
Video over RTP
Server pass-through channel
Video RTP endpoint
External source video over RTP
ADI port
Audio pass-through play filter
Audio RTP endpoint
Audio over RTP
Server pass-through channel
Audio RTP endpoint
External source audio over RTP
Video Access-specific component

To bridge an external RTP source through the Video Messaging Server Interface, the application creates and connects a server pass-through channel, similar to a standard Fusion RTP switching channel. No DSP resources are consumed by the forward operation. For information about a Fusion RTP switching channel, see the *Fusion Developer's Manual*. For information on video specific endpoints and server pass-through channels, see *Creating server pass-through channels* on page 70.

Audio/video bridging is a temporary re-configuration of one of the previous configurations. In the media bridging configuration, play and record from or to the host is not supported.

Initialization function sequence

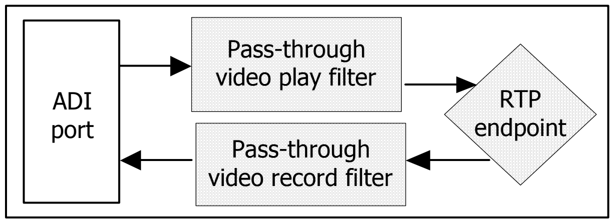
The following table presents a typical function sequence to initialize in a bridging configuration. This sequence contains both video-specific calls and non-video-specific calls.

Note: The bridging configuration is typically initialized in one of the other configurations (for example, pass-through with silence detection).

Step	Call	Description
Forwarding the RTP packets from an external source.		
1	**mspCreateEndpoint** (video_ctahd, &extvideo_ephd)	Creates a video RTP endpoint to receive packets from the external video source.
2	**mspCreateEndpoint** (audio_ctahd, &extaudio_ephd)	Creates an audio RTP endpoint to receive packets from the external audio source.
3	**mspCreateChannel** (video_ctahd, &extvideo_chhd)	Creates a video server pass-through channel (type RTPPassthruSimplex). See *Creating server pass-through channels* on page 70 for more information.
4	**mspCreateChannel** (audio_ctahd, &extaudio_chhd)	Creates an audio server pass-through channel (type RTPPassthruSimplex). See *Creating server pass-through channels* on page 70 for more information.
5	**mspConnect** (video_ephd, extvideo_chhd, extvideo_ephd)	Connects the regular video RTP endpoint, the server pass-through channel, and the external-source-RTP-endpoint together.
6	**mspConnect** (audio_ephd, extaudio_chhd, extaudio_ephd)	Connects the regular audioRTP endpoint, the server pass-through channel, and the external-source-RTP-endpoint together.

Multimedia pass-through video channel operation

This topic describes how the multimedia pass-through video channels operate on the CG board. The pass-through video channel consists of the components shown in the following illustration:


ADI port
Pass-through video play filter
Pass-through video record filter
RTP endpoint

A pass-through channel is created and operated with the following API calls:

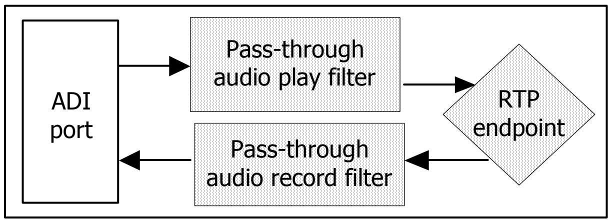
Calling...	Does the following...
ctaOpenServices	• Creates the ADI port. No DSP resource is used.
mspCreateEndpoint	• Creates the video RTP endpoint (to handle MPEG-4, H.263, or H.264).
mspGetFilterHandle	• Translates the MSPP endpoint handle into a CG board endpoint handle that can be used in the ADI service function calls.
adiPlayMMFromMemory or **adiPlayMMAsync**	• Creates the pass-through video play filter. No DSP resource is used. • Connects the ADI port, the pass-through video play filter, and the RTP endpoint. • Starts play of video buffer onto the ADI port.
adiStopPlaying	• Stops the video play. • Disconnects all components. • Destroys the pass-through video play filter.
adiRecordMMToMemory or **adiRecordMMAsync**	• Creates the pass-through video record filter. • Connects the ADI port, the pass-through video record filter, and the RTP endpoint. • Starts recording from the ADI port in the video buffer. • Establishes dependence between video and audio channels due to the audio board endpoint handle provided with the video board endpoint handle.
adiStopRecording	• Stops video record. • Disconnects all components. • Destroys the pass-through video record filter.

Typically, the ADI port and the RTP endpoint exist for the life of the channel. The pass-through video play filter exists for the life of **adiPlayMMFromMemory** or **adiPlayMMAsync**, while the video record filter exists for the life of **adiRecordMMToMemory** or **adiRecordMMAsync**.

It is not necessary for the application to create or destroy the pass-through video filters or for the application to create filter connections or disconnections. These tasks are performed automatically when corresponding ADI service functions are called.

Multimedia pass-through audio channel operation

The pass-through audio channel consists of the components shown in the following illustration:


ADI port
Pass-through audio play filter
Pass-through audio record filter
RTP endpoint

The pass-through audio channel operates as follows:

Calling...	Does the following...
ctaOpenServices	• Creates the ADI port. A DSP resource is used if a timeslot is specified, for example, when silence detection or audio transcoding is needed.
mspCreateEndpoint	• Creates the RTP endpoint (AMR or G.723).
mspGetFilterHandle	• Translates the MSPP endpoint handle into a CG board endpoint handle that can be used in the ADI function calls.
adiPlayMMFromMemory or **adiPlayMMAsync**	• Creates the pass-through audio play filter. No DSP resource is used. • Connects the ADI port, the pass-through audio play filter, and the RTP endpoint together. • Starts play of audio buffer onto the ADI port.
adiStopPlaying	• Stops the audio play. • Disconnects all components. • Destroys the pass-through audio play filter.
adiRecordMMToMemory or **adiRecordMMAsync**	• Creates the pass-through audio record filter. • Connects the ADI port, the pass-through audio record filter, and the RTP endpoint. • Starts recording from the ADI port in the audio buffer. • Establishes inter-dependence between video and audio channels due to the audio board endpoint handle provided with the video board endpoint handle.
adiStopRecording	• Stops the audio record. • Disconnects all components. • Destroys the pass-through audio record filter.

Typically, the ADI port and the RTP endpoint exist for the life of the channel. The pass-through audio play filter exists for the life of **adiPlayMMFromMemory** or **adiPlayMMAsync**, while the pass-through audio record filter exists for the life of **adiRecordMMToMemory or adiRecordMMAsync**.

It is not necessary for the application to create or destroy the pass-through audio filters or for the application to create filter connections or disconnections. These tasks are performed automatically when corresponding ADI service functions are called.

See *Video enhancements to the ADI service* on page 51 and *Video enhancements to the MSPP service* on page 69 for more information.

Configuring endpoints to use audio/video synchronization

You can use RTCP to communicate skew information to the host application and IP destination through RTP endpoints. To use RTCP in this way, enable RTP endpoints as follows:

- Enable RTP receive endpoints to calculate audio and video skew values for incoming data streams and communicate these values to the host application.
- Enable RTP video send endpoints to communicate video skew values to the IP destination.

You can enable RTP endpoints to provide skew information either during endpoint creation or after endpoint creation.

Note: A full-duplex RTCP session is only supported with full-duplex RTP endpoints. A simplex RTP endpoint pair (simplex send endpoint and simplex receive endpoint) does not support a full-duplex RTCP session.

Enabling RTP endpoints to detect and communicate skew values for incoming data streams

Note: This feature is for all audio and video full-duplex and simplex receive RTP endpoints. The example shown is for full-duplex endpoints.

During endpoint creation

When creating an endpoint, you can configure the startRtcp bit field in the RTCP_ENDPOINT_PARMS (or RTCP_V6_ENDPOINT_PARMS) structure so that an endpoint can detect and report skew offset values for incoming data streams to the host application. The endpoint communicates these values by sending MSPEVN_SKEW_OFFSET events to the application.

Use the following macros to configure the startRtcp bit field:

- RTCP_ENABLE to enable RTCP for the endpoint.
- RTCP_ENABLE_RCV_SKEW_CALC to enable the calculation of skew offsets for the endpoint.

You must configure both endpoints in an audio/video stream pair in order to obtain meaningful skew data. For information about using the returned information to calculate audio/video skew, see *Calculating audio/video skew* on page 37.

For more information about creating and configuring RTP endpoints, see RTPRTCP_ENDPOINT_PARMS or RTPRTCP_V6_ENDPOINT_PARMS, the *MSPP Developer's Reference Manual*, and the *Fusion Developer's Manual*.

After endpoint creation

After endpoint creation, you can use the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command against an endpoint so that it can detect and report skew offset values for incoming data streams to the host application. This command uses the msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC structure to indicate whether to enable or disable the skew offset calculation.

You must use the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command against both endpoints in an audio/video stream pair in order to obtain meaningful audio/video skew data. For information, see *Calculating audio/video skew* on page 37.

The MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command can return the following events:

Event	Description
MSPEVN_SENDCOMMAND_DONE	Indicates that the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command was successfully sent to the specified RTP endpoint on the CG board.
MSPEVN_SKEW_OFFSET	Unsolicited event indicating the timing offset (in ms) for the incoming data stream.

For information about creating and sending MSPP commands, see *Creating and sending MSPP commands* on page 72.

Calculating audio/video skew

The following table describes how an application can calculate the audio/video skew value based on information sent from both endpoints in an audio/video stream pair:

Step	Action
1	First endpoint in an audio/video stream pair calculates a skew offset value based on the RTP packet and RTCP sender report timing of an incoming stream.

2	CG board uses an unsolicited MSPEVN_SKEW_OFFSET event to report the skew offset value to the application. The MSPEVN_SKEW_OFFSET event contains the following structure:
	`typedef struct {`
	`DWORD  FilterId;`
	`DWORD  Offset;           //Offset value in ms`
	`} msp_RTP_SKEW_OFFSET;`

3	Second endpoint in the audio/video stream pair calculates a skew offset value based on the RTP packet and RTCP sender report timing of an incoming stream.
4	CG board uses an unsolicited MSPEVN_SKEW_OFFSET event to report the skew offset value to the application.
5	Application subtracts the audio skew offset value from the video skew offset value. This yields the audio/video skew value in ms. A positive result means the video lags the audio, while a negative result means the audio lags the video.
6	Application can do either of the following: • Correct for the incoming audio/video skew on the CG board (possibly by adjusting audio/video jitter buffer latencies). For information about setting the audio jitter buffer latency, see the *MSPP Developer's Reference Manual*. • Use the 3G-324M Interface to signal the audio/video skew value to a receiving 3G-324M terminal through the use of the **h324_h223SkewIndication** function. For more information, see the *3G-324M Interface Developer's Manual*.

Examples

The following example shows how to use the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command so that the server video endpoint with the MSP handle ephd can detect video skew offset values on an incoming data stream:

```
msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC  skewCalc;
skewCalc.enable = RTCP_ENABLE_RCV_SKEW_CALC(0);
//Endian Adjust
skewCalc.enable = H2NMS_DWORD(skewCalc.enable);
command = mspBuildCommand(msp_ENDPOINT_RTPFDX_VIDEO,
                          MSP_CMD_RTPFDX_CALC_SKEW_OFFSET);
ret = mspSendCommand(ephd, command, &skewCalc, sizeof(skewCalc));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
                 MSP_CMD_RTPFDX_CALC_SKEW_OFFSET);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

The following example shows how to use the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command so that the audio endpoint with the MSP handle ephd1 can detect audio skew offset values on an incoming data stream:

```
msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC  skewCalc;
skewCalc.enable = RTCP_ENABLE_RCV_SKEW_CALC(0);
//Endian Adjust
skewCalc.enable = H2NMS_DWORD(skewCalc.enable);
command = mspBuildCommand(msp_ENDPOINT_RTPFDX),
                          MSP_CMD_RTPFDX_CALC_SKEW_OFFSET);
ret = mspSendCommand(ephd1, command, &skewCalc, sizeof(skewCalc));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
                 MSP_CMD_RTPFDX_CALC_SKEW_OFFSET);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

The following example shows how to calculate the audio/video skew value upon receipt of MSPEVN_SKEW_OFFSET events from the audio data stream and the video data stream:

```
switch (pevent->id)
{
   case MSPEVN_SKEW_OFFSET:
      msp_RTP_SKEW_OFFSET *pSkewEvent;
      pSkewEvent = (msp_RTP_SKEW_OFFSET*)(pevent->buffer);
      skewOffset = (int)NMS2H_DWORD(pSkewEvent->offset);
      if(VideoCtx[nGw].rtpEp.hd == pevent->objHd)
      {
         vidOffset = skewOffset;
         bVid = TRUE;
      }
      else
      {
         audOffset = skewOffset;
         bAud = TRUE;
      }
      if (bVid == TRUE && bAud == TRUE)
      {
         vidSkew = vidOffset - audOffset;
         printf("\nVideo lags audio by %d ms", vidSkew);
         bVid = bAud = FALSE;
      }
      break;
}
```

Enabling RTP endpoints to send video skew values to the IP destination

Note: This feature is for server full-duplex video RTP endpoints.

During endpoint creation

When creating a video endpoint, you can configure the startRtcp bit field in the RTCP_ENDPOINT_PARMS (or RTCP_V6_ENDPOINT_PARMS) structure so that the endpoint can send video skew values to the IP destination. The configured endpoint sends video skew values in RTCP Sender Reports.

Use the following macros to configure the startRtcp bit field:

- RTCP_ENABLE to enable RTCP for the endpoint.
- RTCP_SET_0_INTERVAL to determine how quickly the RCTP Sender Reports begin after the RTP stream begins.
- RTCP_VIDEO_LEADS_AUDIO, to indicate that video leads audio (if true).

For more information about creating and configuring RTP endpoints, see *RTPRTCP_ENDPOINT_PARMS* on page 76 or RTPRTCP_V6_ENDPOINT_PARMS, the *MSPP Developer's Reference Manual*, and the *Fusion Developer's Manual*.

After endpoint creation

After endpoint creation, you can use the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command against the video endpoint so that the endpoint sends video skew values to the IP destination. The configured endpoint sends video skew values in RTCP Sender Reports. The MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command uses the msp_ENDPOINT_RTPFDX_SET_VID_SKEW structure to set the skew value.

The MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command can return the following event:

Event	Description
MSPEVN_SENDCOMMAND_DONE	Indicates that the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command was successfully sent to the specified endpoint.

For information about creating and sending MSPP commands, see *Creating and sending MSPP commands* on page 72.

Example

The following example shows how to use the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command against the video endpoint with the MSP handle ephd so that the endpoint can signal video skew to the IP destination:

```
msp_ENDPOINT_RTPFDX_SET_VID_SKEW  skewCmd;


skewCmd.vidSkew = 0;

if (skew < 0)
{
   //Video leads audio
   skewCmd.vidSkew = RTCP_VIDEO_LEADS_AUDIO(skewCmd.vidSkew);
   skew *= -1;  //Make skew a positive number
}
skewCmd.vidSkew = RTCP_VIDEO_SKEW(skewCmd.vidSkew, skew);
//Endian Adjust
skewCmd.vidSkew = H2NMS_DWORD(skewCmd.vidSkew);
command = mspBuildCommand(msp_ENDPOINT_RTPFDX_VIDEO,
                          MSP_CMD_RTPFDX_VIDEO_SKEW_TIME);
ret = mspSendCommand(ephd, command, &skewCmd, sizeof(skewCmd));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
                 MSP_CMD_RTPFDX_VIDEO_SKEW_TIME);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

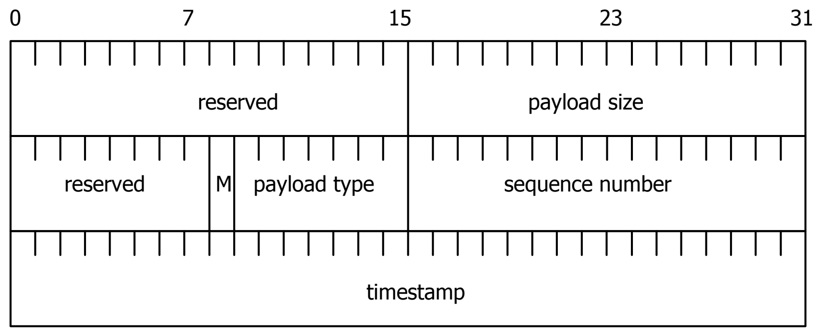
4 Hosting media buffering and formatting

Formatting media buffers for pass-through channels

The proprietary NMS packetized format defines the structure of the data passed between the application and the ADI multimedia functions on play and record for pass-through channels.

Operation	Description
Play	The application reads the audio/video files from disk, loads the streams onto host memory, and calls **adiPlayMMFromMemory** or **adiPlayMMAsync** one time each for video and audio, using the corresponding buffer sizes and pointers.
Record	The application allocates two host memory buffers to receive the complete recorded streams, and calls **adiRecordMMToMemory** or **adiRecordMMAsync** one time each for video and audio, using the corresponding buffer sizes and pointers.

The data within the audio and video buffers consist of a sequence of packets arranged in the order in which they were received (recorded) or to be delivered (played). The packet header is referred to as the NMS packetized header (and is similar to an RTP header):



The second reserved field (8 bits) is the NMS magic number and is set to 0xD.

Timestamps

For audio channels, the timestamps are 8000/sec. This originates from the standard 8 kHz audio sampling rate.

For video channels, the timestamps are 90000/sec. This originates from the standard 90 kHz video sampling rate.

Payload size

For video NMS headers, the payload size is not constant because the video bit rate is variable.

For audio NMS headers, the payload size depends on the coding rate chosen for AMR or G.723 encoding, or whether G.711 is being used.

It may not be constant if the DTX function (voice activity detection + comfort noise generation) is enabled on the audio encoder. In this case, when silence is detected, the audio encoder can generate non-speech frames, such as Silence Insertion Descriptor (SID) frames.

Payload type

Payload types qualify the media encoding of the payload, comply with the standard profiles, and use by default, the values defined by RFC 3551, where applicable. During a play operation RTP packets are transmitted with the payload type value taken directly from the NMS-packetized header.

When the audio and video buffers are generated by the Multimedia File Interface library functions, the application can control the values of payload types.

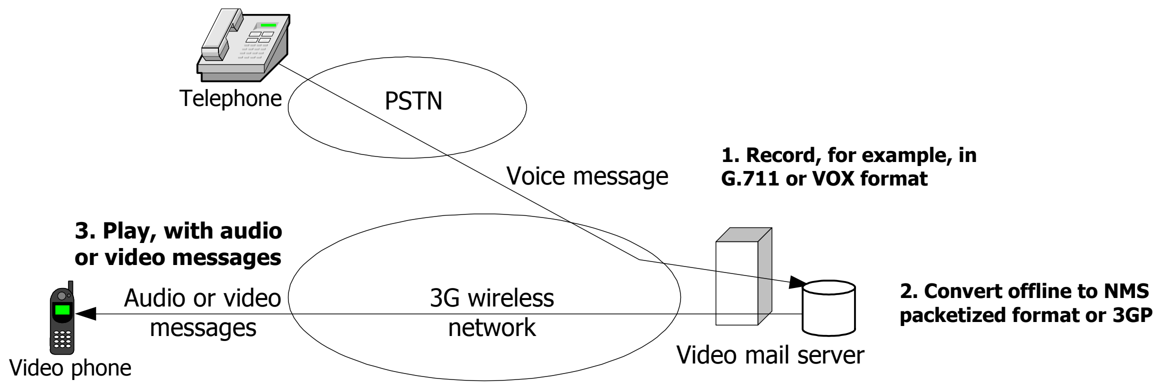
Payload content

The RTP payload for each RTP packet is located after the NMS header. The RTP payload can contain a media specific header:

- For each H.263 RTP packet, the RTP payload consists of the H.263 payload header (either per RFC 2190 or RFC 2429) followed by the H.263 compressed bit stream.
- For each H.264 RTP packet, the RTP payload does not contain a media specific header, but contains the H.264 compressed bit stream only.
- For each MPEG-4 RTP packet, the RTP payload does not contain a media specific header, but contains the MPEG-4 compressed bit stream only.
- For audio the payload contains only the audio bit stream.

Converting audio offline

While it is possible to convert between audio formats using the audio transcoding configuration of the Video Messaging Server, it may be necessary to perform offline conversions of pre-recorded audio to 3GP audio. The following illustration shows the offline conversion process:

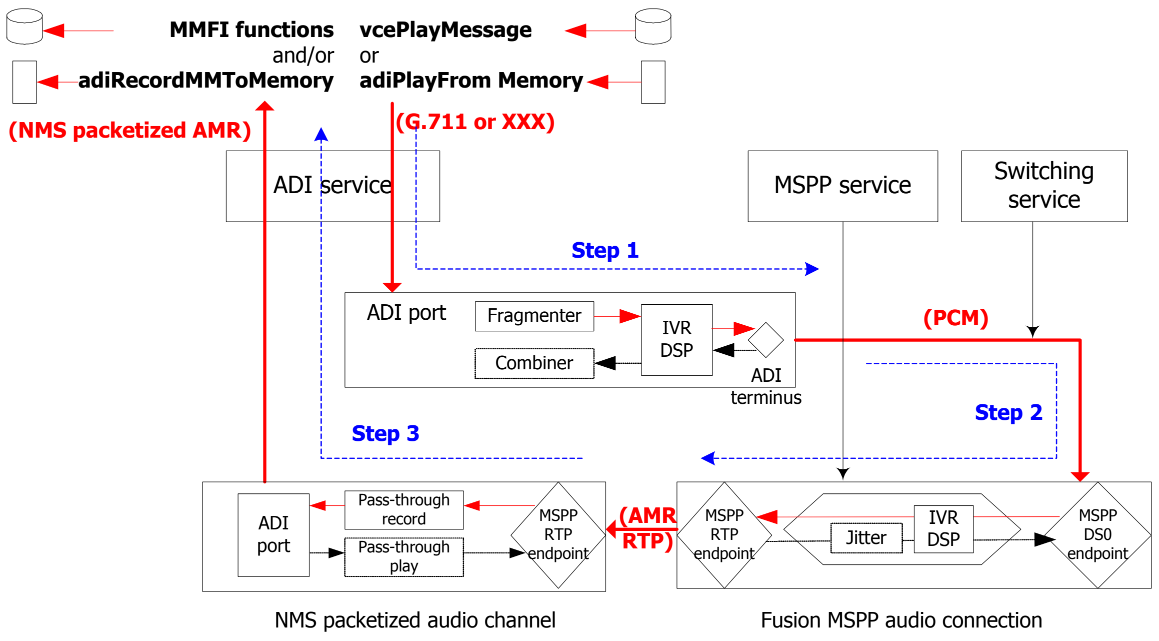

Telephone
PSTN
Voice message
1. Record, for example, in G.711 or VOX format
3. Play, with audio or video messages
Audio or video messages
3G wireless network
2. Convert offline to NMS packetized format or 3GP
Video mail server
Video phone

This topic provides guidelines for using Video Access to complete the following offline conversions:

- Regular voice to 3GP audio
- 3GP audio to regular voice

Regular voice to 3GP audio offline conversion

Converting regular voice to 3GP audio uses memory play/record APIs. The following illustration shows these steps:



Step	Description
1	The voice message is played and decoded with the ADI service or the Voice Message service as if it were played onto a telephone circuit. Assuming the voice message was recorded in an encoding format supported by the ADI service, it can be played back with an ADI service play function (for example, **adiPlayFromMemory** or **adiPlayAsync**) or with a Voice Message service play function (for example, **vcePlayMessage**). In both cases, the play operation is performed on a DSP port of a CG board, which decodes the data and generates a PCM stream. For more information, see the *ADI Service Developer's Reference Manual* and the *Voice Message Service Developer's Reference Manual*.
2	The generated PCM stream is re-encoded and packetized into AMR RTP packets with the MSPP service. Instead of switching the output of the decoder onto a T1/E1 timeslot, use the Switching service to switch it to the DS0 endpoint of an MSPP AMR connection (for example, the set of a DS0 endpoint, an AMR channel, and an RTP endpoint). The PCM audio stream is AMR-encoded and RTP-packetized as for a typical Fusion gateway operation. Configure the destination address of the RTP endpoint of that MSPP connection so that packets are sent to the audio RTP endpoint created in Step 3. For more information, see the *MSPP Service Developer's Reference Manual* and the *Switching Service Developer's Reference Manual*.
3	The AMR RTP stream is recorded in an NMS packetized format with the ADI service. Create an RTP audio endpoint and configure its source address to receive RTP packets from the MSPP connection previously created. For more information, see *Video enhancements to the MSPP service* on page 69. Record the received AMR RTP stream in the NMS packetized format with **adiRecordMMToMemory** or **adiRecordMMAsync**. Use Multimedia File library functions to store the recorded message into a 3GP file.

3GP audio to regular voice offline conversion

Converting 3GP audio to regular voice uses the same path, but in the reverse direction:

Step	Description
1	The 3GP audio message is played in the NMS packetized format with **adiPlayMMFromMemory** or **adiPlayMMAsync** (preceded with calls to Multimedia File library functions, if the message was originally stored in a 3GP file). An AMR RTP stream is generated out of an RTP audio endpoint whose destination IP address is that of the receiving RTP endpoint in Step 2.
2	The AMR RTP stream is received by an MSPP RTP endpoint, decoded through an MSPP AMR channel, and turned into a PCM audio stream out of an MSPP DS0 endpoint.
3	The PCM stream is switched to a DSP port and recorded with an ADI service function or a Voice Message service function using the desired encoding format.

3GP file compatibility

With the Multimedia File Interface library, video messaging applications can record audio/video messages into 3GP files. Very often, those messaging applications need to play back several 3GP files in sequence, including pre-recorded prompts, service information, announcements, and so on.

Creating these 3GP files requires specific offline recording and editing tools. In addition, these files must satisfy certain rules to ensure they can be played in sequence with files recorded with the 3GP library.

Creating and editing 3GP files

There are multiple tools available that can play, record, edit, import, and export audio/video files in 3GP format. It is beyond the scope of Video Access for NMS to compare them or recommend a particular tool. NMS has validated that 3GP files generated with the Video Access 3GP library are compatible with several tools available on the market. Using audio and video, messages can be prepared offline, stored into 3GP format files, and then played with Video Access. 3GP files can be created with AMR NB audio encoding, or MPEG-4, H.263, or H.264 video encoding.

For compatibility with NMS Video Access, audio encoding must be AMR NB. NMS recommends a default rate of 12.2 kbit/s.

When the target terminal is a 3G-324M device, video encoding can be H.263, H.264, or MPEG-4. In all cases, the data rate must be capped at 43 kbit/s maximum to guarantee that a 64 kbit/s 3G-324M channel can accommodate both media streams. QCIF resolution should be used, and frame rate should not exceed 15 fps.

Playing a sequence of 3GP files

When playing a sequence of 3GP files towards a 3G-324M terminal, it is possible that those files do not have the same video encoding. For example, some may be H.263, some H.264, and others MPEG-4.

As most terminals can support a single video encoding during a call, messaging systems must either prepare the sequence of files offline in all formats before the call, or insert online an NMS video transcoder channel when needed. The video transcoder channel presently supports transcoding between H.263 and MPEG-4 only.

The first method adapts well to generic messages such as prompts or service information, which are typically created with an editing tool. The second method adapts to recorded messages, for example, messages left in a video mailbox.

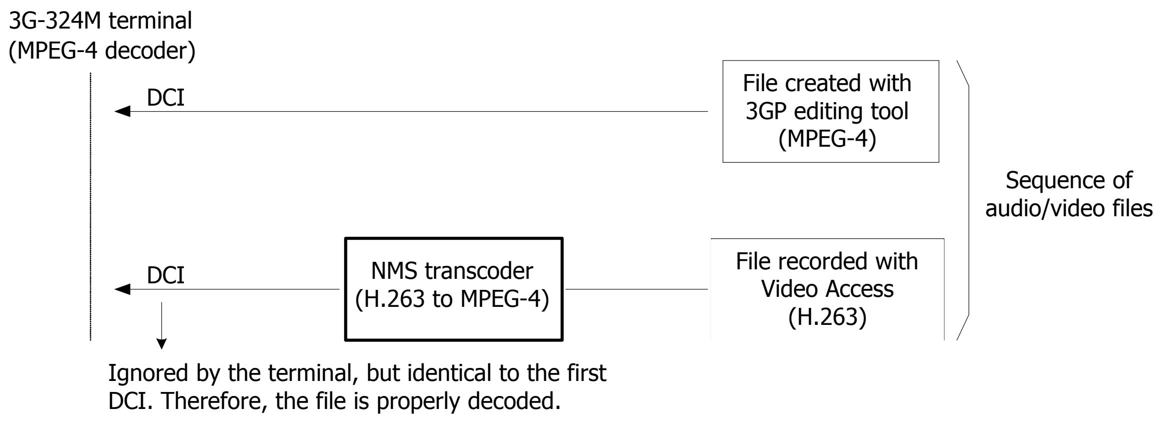
Therefore, when several 3GP files are played in sequence, the following types of files can be encountered:

- Files pre-recorded or edited with a 3GP editing tool, in the same video encoding as the terminal capability.
- Files recorded with NMS Video Access in the same video encoding as the terminal capability.
- Files recorded with NMS Video Access in a different video encoding than the terminal capability.

MPEG-4 VO/VOL

MPEG-4 video encoding requires special attention. By definition, the decoder configuration information (DCI) used by the receiver to decode the video stream is carried in the video bit stream itself. Consequently, when playing a sequence of files, several VO/VOL (Video Object/Video Object Layer) information blocks are transmitted to the terminal.

Terminals may not support dynamic reconfiguration of their video decoder based on in-band DCI change. They consider the first DCI received (typically duplicated with out-of-band H.245 signaling) and ignore all subsequent DCI blocks. It is critical to ensure that all files in a sequence have the same DCI, so that the first DCI received is valid for all messages. The following illustration shows an example of this.


3G-324M terminal
(MPEG-4 decoder)
DCI
File created with 3GP editing tool (MPEG-4)
Sequence of audio/video files
DCI
NMS transcoder (H.263 to MPEG-4)
File recorded with Video Access (H.263)
Ignored by the terminal, but identical to the first DCI. Therefore, the file is properly decoded.

When the target terminal is a 3G-324M device, another option to play in sequence 3GP MPEG4 clips encoded with different DCI is to close the video logical channel and then re-open the channel using the matching DCI. You must ensure that the 3G-324M terminal supports this capability. Refer to the *3G-324M Interface Developer's Reference Manual* for information.

H.264 DCI

Similarly to MPEG-4, an H.264 decoder must be configured with decoder configuration information, which concatenates H.264 sequence parameter sets (SPS) and picture parameter sets (PPS).

Because 3G-324M terminals may not support dynamic reconfiguration of their H.264 video decoder based on in-band DCI change, Video Access provides the capability to signal out-of-band (using H.245 signaling) H.264 DCI changes. This way, the application can play in sequence 3GP H.264 clips encoded with different DCI. Refer to the *3G-324M Interface Developer's Reference Manual* for more information.

5 ADI service

ADI service

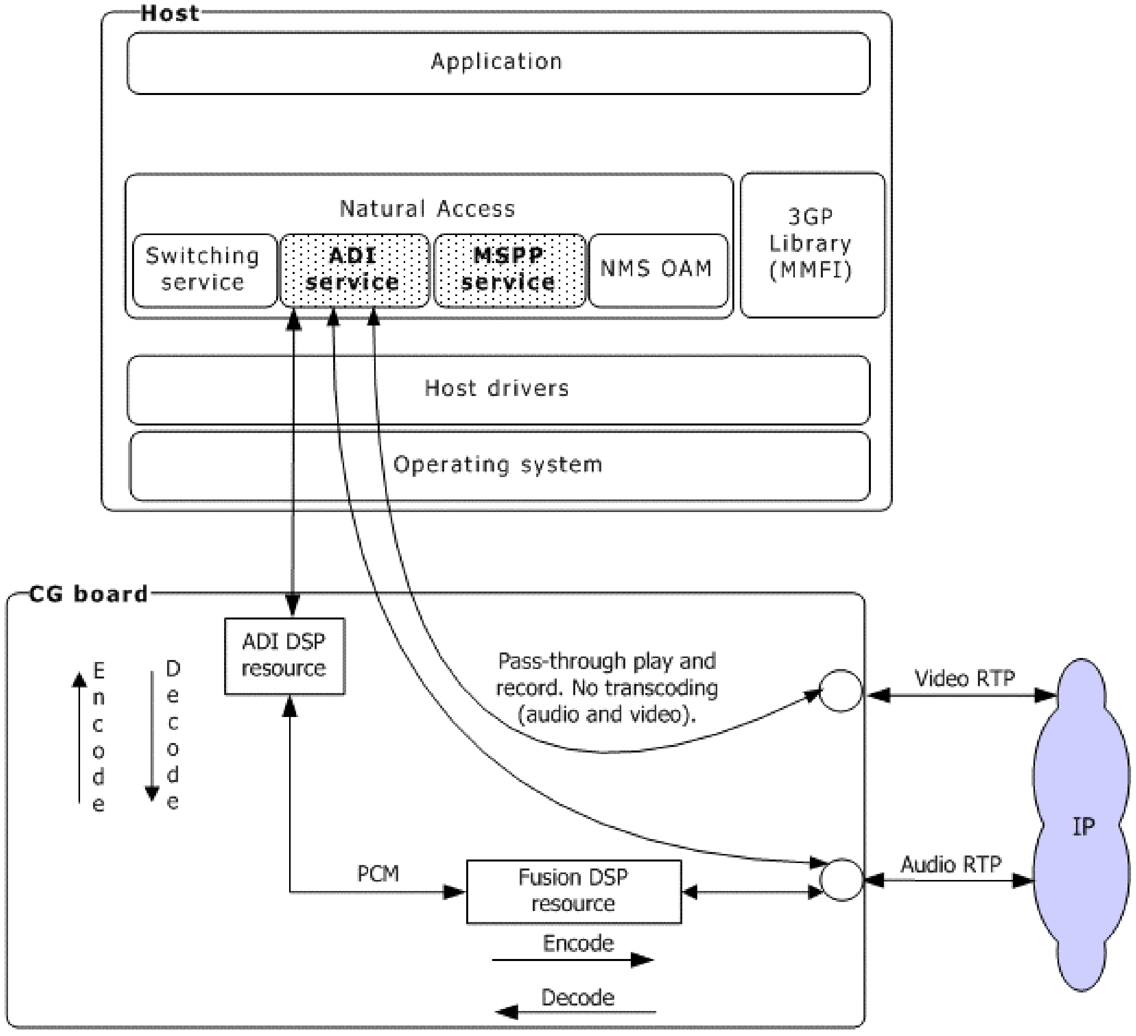
The application accesses multimedia play and record functions through the ADI service. It can use **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, and **adiRecordMMAsync** for both audio and video media. These functions allow the application to play and record:

- Audio and video data directly to or from the RTP endpoints respectively (pass-through configuration). In this configuration, the media format at the host for all functions is NMS packetized format.
- Video data directly to or from the RTP endpoints while transcoding the audio data (audio transcoding configuration). In this configuration, the audio media format at the host for all functions is raw audio bit streams, while the video media format is NMS packetized format.

For more information, see *Video enhancements to the ADI service* on page 51.

Note: In the Video Messaging Server Interface header files, pass-through channels are referred to as native channels.

If audio transcoding is required, the application uses one of the ADI service multimedia play/record functions to play and record to an existing Fusion channel. In this case, the multimedia ADI function specifies the encoding type at the host while an MSPP service function specifies the audio encoding type for the Fusion channel that terminates the IP network. The ADI service and the MSPP service use DSP resources on the board as shown in the following illustration.



Video enhancements to the ADI service

The Video Messaging Server Interface enhances the Natural Access ADI service to support video applications and allow pass-through RTP play and record functions. The ADI service supports encoding types for audio and video channels.

The multimedia functions **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, and **adiRecordMMAsync** can control audio and video streams in the following ways:

- Start playing audio or video streams
- Stop playing audio or video streams
- Start recording audio or video streams
- Simultaneously stop recording audio and video streams
- Simultaneously perform play and record operations on the same context
- Automatically stop recording on audio silence detection
- Automatically stop recording on video silence detection (no video decoding is performed)

These functions are based on standard ADI functions, as shown in the following table:

Enhanced ADI function	Associated standard ADI function
adiPlayMMFromMemory	**adiPlayFromMemory**
adiPlayMMAsync	**adiPlayAsync**
adiRecordMMToMemory	**adiRecordToMemory**
adiRecordMMAsync	**adiRecordAsync**

For information about the standard ADI play and record commands, see the *ADI Service Developer's Reference Manual.*

Note: The MM in the video play and record functions stands for MultiMedia.

Pass-through play and record limitations

The pass-through play and record limitations are the following:

- When DTMF is transported and detected out-of-band, automatic stop of record operations on DTMF is not supported. Typically, in the context of 3G-324M communications, user interaction is carried in H.245 UII message indications, and relayed out-of-band by the 3G-324M Interface to the server. See the *3G-324M Interface Developer's Reference Manual* for more information. The application can stop recording but automatic record stop on H.245 UII message reception is not supported.
- Existing ADI record parameters such as AGC and speed are not supported by **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, and **adiRecordMMAsync**.

- Other ADI audio processing functions such as tone detection or echo cancellation cannot be executed on a context that was opened for play or record pass-through multimedia streams. Only start multimedia record (**adiRecordMMToMemory** or **adiRecordMMAsync**), start multimedia play (**adiPlayMMFromMemory** or **adiPlayMMAsync**), stop record (**adiStopRecording**), and stop play (**adiStopPlaying**) are supported.
- Record and play operations do not necessarily require an audio port and a video port. For example, audio content can be played while no associated video is played.
- When video I-frame synchronization is turned on, the multimedia video record function must be started before the multimedia audio record function, even when using **adiRecordMMToMemory** or **adiRecordMMAsync** for audio transcoding. In other cases of play and record, the order does not matter, and either the audio or the video function can be started first.

Stop functions

Use the following standard ADI service functions to stop playing (**adiStopPlaying**) and to stop recording (**adiStopRecording**) pass-through multimedia streams:

DWORD **adiStopPlaying** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

DWORD **adiStopRecording** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

When a record operation is stopped on a context (audio or video), record is automatically stopped on the other context in the pass-through configuration and in the audio transcoding configuration when using either **adiRecordMMToMemory** or **adiRecordMMAsync**. See *Defining the Video Messaging Server configuration* on page 19 for more information.

When a play operation is stopped on a context, it is not automatically stopped on the other context.

Multimedia encoding types

The ADI service supports the following encoding types to support the multimedia pass-through play/record features:

Media function	Encoding type
Pass-through audio	ADI_ENCODE_NATIVE_AMRNB ADI_ENCODE_NATIVE_G_723_1 ADI_ENCODE_NATIVE_G_711
Pass-through video	ADI_ENCODE_NATIVE_MPEG4 ADI_ENCODE_NATIVE_H_263 (RFC 2190) ADI_ENCODE_NATIVE_H_263P (RFC 2429) ADI_ENCODE_NATIVE_H_264

Specify these encoding types in **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, and **adiRecordMMAsync**.

Note: When performing audio transcoding, any ADI-supported audio encoding type can be used. However, when using 3GP files, an AMR encoding type must be used (ADI_ENCODE_AMR_***), where *** is based on the frame rate. For more information, see the *ADI Service Developer's Reference Manual*.

adiPlayMMAsync

Initiates a multimedia play operation with asynchronous buffer submission.

Prototype

DWORD **adiPlayMMAsync** (CTAHD ***ctahd***, unsigned ***encoding***, void ****buffer1***, unsigned ***bufsize***, unsigned ***flags***, DWORD ***ephd***, ADI_PLAY_PARMS ****parms***);

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding type. See *adidef.h* header file for a complete list of ADI_ENCODING_***xxx*** values. See *Multimedia encoding types* on page 53 for the RTP compressed format encoding types.
buffer1	Pointer to the initial media data buffer.
bufsize	Number of bytes pointed to by ***buffer1***.
flags	Indicates if the specified buffer is the only buffer to be played. Valid values are: • ADI_PLAY_LAST_BUFFER - Specified buffer is the only buffer to be played. • 0 - There are subsequent buffers to play.
ephd	Pointer to the CG board filter endpoint handle obtained with **mspGetFilterHandle**. This endpoint handle is required to connect the ADI channel on the board to the RTP endpoint.
parms	Specify a NULL pointer. Standard ADI play parameters are not applicable for multimedia pass-through play functions.

Return values

Return value	Description
SUCCESS	
ADIERR_INVALID_CALL_STATE	Function not valid in the current call state.
CTAERR_BAD_ARGUMENT	Either invalid ***encoding*** or NULL ***buffer1***.
CTAERR_FUNCTION_ACTIVE	Function already started.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_STATE	Function not valid in the current port state.
CTAERR_OUTPUT_ACTIVE	Play failed because there is another active output function.
CTAERR_SVR_COMM	Server communication error.

Events

Event	Description
ADIEVN_PLAY_BUFFER_REQ	Generated when the ADI service needs a buffer containing multimedia data. The application responds by either submitting a full buffer (**adiSubmitPlayBuffer**) or a full or partial buffer (**adiSubmitPlayBuffer** with a flag indicating ADI_PLAY_LAST_BUFFER). If the ADI_PLAY_UNDERRUN bit is set, an underrun occurred, meaning that playing was temporarily suspended because there was no buffer to play.
ADIEVN_PLAY_DONE	Generated by the ADI service when the play operation terminates. The event size field contains the total number of bytes played during the function's instance. The event value field contains one of the following termination conditions, or an error code: CTA_REASON_DIGIT Aborted due to DTMF. CTA_REASON_FINISHED Buffer submitted with the ADI_PLAY_LAST_BUFFER flag set completed playing. CTA_REASON_RELEASED Call terminated. CTA_REASON_STOPPED Stopped by application request.

Details

Use **adiPlayMMAsync** to initiate a multimedia playback operation. The multimedia data is supplied in a sequence of buffers. The application has complete responsibility for allocating, filling, and submitting buffers to the ADI service.

The ***bufsize*** should be set to a value less than or equal to the board buffer size, which by default is 16,400 for CG boards. In this case, you can reuse the buffer as soon as this function returns.

After play initiates, the ADI service sends the ADIEVN_PLAY_BUFFER_REQ event to the application whenever it is ready to process more data. The application responds to this event by submitting a filled voice buffer with **adiSubmitPlayBuffer**. The application must submit buffers only in response to ADIEVN_PLAY_BUFFER_REQ.

For proper operation, each buffer must be submitted while the previous buffer is being played.

The application terminates play by submitting a buffer with the flags argument set to ADI_PLAY_LAST_BUFFER. After the ADI service has played the buffer that was submitted with the flag set, it generates an ADIEVN_PLAY_DONE event with the value set to CTA_REASON_FINISHED. To avoid unintentionally modifying data, the application must not modify the buffer until it receives the DONE event.

For more information about **adiSubmitPlayBuffer**, see the *ADI Service Developer's Reference Manual*.

See also

adiRecordMMAsync

Example

```
ret = adiPlayMMAsync(VideoCtx[i].ctahd,
      mmParm.ADIvidEncoder, // Encoding
      VideoCtx[i].data_buffer, // Buffer to play data from
      16400, // data size to play
      0, // flags - not last buffer
      videoEP_filterhd, // Video EndPoint Filter Handle
      NULL);
if (ret != SUCCESS)
{
    printf("Failed to Play Async, %x\n", ret);
    exit(1);
}
```

adiPlayMMFromMemory

Initiates a multimedia play operation using data from a single memory-resident buffer.

Prototype

DWORD **adiPlayMMFromMemory** (CTAHD ***ctahd***, unsigned ***encoding***, void ****buffer***, unsigned ***bufsize***, MSPHD ****ephd***, ADI_PLAY_PARMS ****parms***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding type. See *adidef.h* for a complete list of ADI_ENCODING_xxx values. See *Multimedia encoding types* on page 53 for the RTP compressed format encoding types.
buffer	Pointer to the media data buffer.
bufsize	Number of bytes stored at the address in ***buffer*** (***bufsize*** can be arbitrarily large).
ephd	Pointer to the CG board filter endpoint handle from **mspGetFilterHandle**. This endpoint handle is required to connect the ADI channel on the board to the RTP endpoint.
parms	Specify a NULL pointer. Standard ADI play parameters are not applicable for multimedia pass-through play functions.

Return values

Return value	Description
SUCCESS	
ADIERR_INVALID_CALL_STATE	Function not valid in the current call state.
CTAERR_BAD_ARGUMENT	Either invalid ***encoding*** or NULL ***buffer***.
CTAERR_FUNCTION_ACTIVE	Function already started.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_STATE	Function not valid in the current port state.
CTAERR_OUTPUT_ACTIVE	Play failed because there is another active output function.
CTAERR_SVR_COMM	Play or record communication error.

Events

The ADIEVN_PLAY_DONE event is generated by the ADI service when playing terminates. The event size field contains the total number of bytes played during the function instance. The event value field contains one of the following termination reasons or an error code:

Event	Description
CTA_REASON_DIGIT	Aborted due to DTMF.
CTA_REASON_FINISHED	Complete buffer played.
CTA_REASON_RELEASED	Call terminated.
CTA_REASON_STOPPED	Stopped by application request.

Details

adiPlayMMFromMemory starts playing a single memory-resident buffer of ***bufsize*** bytes. The ADI service generates ADIEVN_PLAY_DONE when the function terminates. To avoid unintentionally modifying data, the application must not modify the buffer until it receives the DONE event.

See also

adiRecordMMToMemory

Example

```
if ((VideoCtx[i].video_fp = fopen(video_FileName, "r")) == NULL)
{
printf("Failed to open file: %s\n", video_FileName);
exit(1);
}
wrSize = fread(VideoCtx[i].data_buffer,
sizeof(BYTE),
mmParm.video_buffer_size,
VideoCtx[i].video_fp);

ret = adiPlayMMFromMemory(VideoCtx[i].ctahd,
mmParm.ADIvidEncoder, // Encoding
VideoCtx[i].data_buffer, // Buffer to play data from
wrSize, // data size to play
videoEP_filterhd, // Video EndPoint Filter Handle
NULL);
if (ret != SUCCESS)
{
printf("Failed to Play from memory, %x\n", ret);
exit(1);
}
```

adiRecordMMAsync

Initiates multimedia recording in asynchronous buffer mode. In this mode, the ADI service generates a buffer full event when each buffer is full. The application asynchronously stores the data and submits empty buffers in response.

Parameters to control silence detection are provided and apply if silence detection is configured.

Prototype

DWORD **adiRecordMMAsync** (CTAHD ***ctahd***, unsigned ***encoding***, void ****buffer1***, unsigned ***bufsize***, DWORD ****video_ephd***, DWORD ****audio_ephd***, ADI_MM_RECORD_PARMS ****parms***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding type. See *adidef.h* for a complete list of ADI_ENCODING_xxx values. See *Multimedia encoding types* on page 53 for the RTP compressed format encoding types.
buffer1	Pointer to the media data buffer.
bufsize	Number of bytes pointed to by ***buffer1***.
video_ephd	Pointer to the CG board video filter endpoint handle obtained with **mspGetFilterHandle**. The application uses this endpoint handle to connect the ADI channel on the board to the RTP endpoint and to synchronize the beginning and ending of the audio and video record.
audio_ephd	Pointer to the CG board audio filter endpoint handle obtained with **mspGetFilterHandle**. The application uses this endpoint handle to connect the ADI channel on the board to the RTP endpoint and to synchronize the beginning and ending of the audio and video record.
parms	Pointer to the ADI_MM_RECORD_PARMS structure that contains record parameters.

Return values

Return value	Description
SUCCESS	
ADIERR_INVALID_CALL_STATE	Function not valid in the current call state.
CTAERR_BAD_ARGUMENT	Either invalid ***encoding*** selected or NULL ***buffer1*** pointer passed.
CTAERR_FUNCTION_ACTIVE	Function already started.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_STATE	Function not valid in the current port state.
CTAERR_OUTPUT_ACTIVE	Record failed because there is another active output function.
CTAERR_RESOURCE_CONFLICT	Silence detector is in use by **adiStartEnergyDetector**.
CTAERR_SVR_COMM	Server communication error.

Events

Event	Description
ADIEVN_RECORD_DONE	Generated when the record operation completes. The event size field contains the total number of bytes recorded during the record instance lifetime. The value field contains a reason code or an error code. The reason codes are described in the Reason codes for ADIEVN_RECORD_DONE table.
ADIEVN_RECORD_BUFFER_FULL	Generated by the ADI service when a buffer is filled with recorded voice data. The event contains the following fields: • buffer - Pointer to a previously submitted user ***buffer1***. • size - Number of bytes recorded into the buffer. • value - Flags. If the ADI_RECORD_BUFFER_REQ bit is set, more buffers are needed and the application must submit another empty buffer. If the ADI_RECORD_UNDERRUN bit is set, an underrun occurred. There was no new buffer to record information when this one was completed.
ADIEVN_RECORD_STARTED	Generated by the ADI service after the function is started on the board. If the ADI_RECORD_BUFFER_REQ bit in the event value field is set, more buffers are needed and the application must submit another empty buffer.

Reason codes for ADIEVN_RECORD_DONE

Reason	Description
CTA_REASON_FINISHED	Buffer is filled.
CTA_REASON_MAXTIME_AUDIO	Maximum recording time reached. Applicable for pass-through audio encoding types only.
CTA_REASON_MAXTIME_VIDEO	Maximum recording time reached. Applicable for pass-through video encoding types only.
CTA_REASON_NATIVE_COMPANION_RECORD_STOPPED	Companion record channel stopped. For example, a pass-through audio channel and a pass-through video channel are companion channels because they have matching video_ephd and audio_ephd input parameters. If one of the two channels stops recording for any reason, the companion pass-through channel is also stopped with this reason code.
CTA_REASON_NO_VIDEO	No video detected.
CTA_REASON_NO_VOICE	No voice detected.
CTA_REASON_RELEASED	Call terminated.
CTA_REASON_STOPPED	Stopped by application request.
CTA_REASON_VIDEO_END	Video packets not detected for specified timeout period. Applicable only for pass-through video encoding types.
CTA_REASON_VOICE_END	User stopped speaking.
CTAERR_FUNCTION_NOT_AVAIL	Required DSP file not loaded on the board.
CTAERR_xxx or ADIERR_xxx	Record failed.

Details

Use **adiRecordMMAsync** to initiate a multimedia record operation. The data is supplied to the application in a sequence of buffers. The application submits empty buffers using **adiSubmitRecordBuffer** for the duration of the operation. These buffers are then filled with recorded voice data and ADIEVN_RECORD_BUFFER_FULL events are returned. The application is responsible for allocating, flushing, and submitting the buffers.

When the ADI service needs another buffer, it sets the ADI_RECORD_BUFFER_REQ bit in the event value field for ADIEVN_RECORD_STARTED and ADIEVN_RECORD_BUFFER_FULL. The application responds by submitting another empty buffer using **adiSubmitRecordBuffer**. The application submits buffers only when requested by the ADI service. The ADI service owns the buffer until either ADIEVN_RECORD_BUFFER_FULL or ADIEVEN_RECORD_DONE is delivered to the application.

The last buffer before the DONE event can be a partial buffer. The DONE event itself does not include a buffer of data. The record operation terminates when the application receives ADIEVN_RECORD_DONE.

Note: The final buffer that was submitted may not be returned to the application. If the application dynamically allocates buffers, it must keep track of submitted buffers to free any outstanding buffers when record is done.

The ***bufsize*** should be set to the board buffer size, which by default is 16,400 for CG boards.

For proper operation, each buffer must be submitted while the previous buffer is being filled.

For more information about **adiSubmitRecordBuffer**, see the *ADI Service Developer's Reference Manual*.

Example

```
if ((ret = SetRecordParms(VideoCtx[i].ctahd, &recParms)) != SUCCESS)
{
    printf("Failed to setup Record Parameters\n");
    exit(1);
}
ret = adiRecordMMAsync(VideoCtx[i].ctahd,
mmParm.ADIvidEncoder,      // Encoding
VideoCtx[i].data_buffer,   // Buffer to receive recorded video
16400,                     // Buffer size
videoEP_filterhd,          // Video EndPoint Filter Handle
audioEP_filterhd,          // Audio EndPoint Filter Handle
&recParms);                // Record parameters
if (ret != SUCCESS)
{
    printf("Failed to Record Async, %x\n", ret);
    exit(1);
}
```

adiRecordMMToMemory

Initiates multimedia recording to a memory-resident buffer of a specified size and returns to the application. The ADI service records data into the buffer until one of the terminating conditions described in the ADIEVN_RECORDING_DONE event occurs.

Parameters to control silence detection are provided and apply if silence detection is configured.

Prototype

DWORD **adiRecordMMToMemory** (CTAHD ***ctahd***, unsigned ***encoding***, void ****buffer***, unsigned ***bufsize***, MSPHD ****video_ephd***, MSPHD ****audio_ephd***, ADI_MM_RECORD_PARMS ****parms***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
encoding	Encoding type. See the *adidef.h* header file for a complete list of ADI_ENCODING_***xxx*** values. See *Multimedia encoding types* on page 53 for the RTP compressed format encoding types.
buffer	Pointer to the media data buffer.
bufsize	Number of bytes pointed to by ***buffer***.
video_ephd	Pointer to the CG board video filter endpoint handle obtained with **mspGetFilterHandle**. The application uses this endpoint handle to connect the ADI channel on the board to the RTP endpoint and to synchronize the beginning and ending of the audio and video record.
audio_ephd	Pointer to the CG board audio filter endpoint handle obtained with **mspGetFilterHandle**. The application uses this endpoint handle to connect the ADI channel on the board to the RTP endpoint and to synchronize the beginning and ending of the audio and video record.
parms	Pointer to the ADI_MM_RECORD_PARMS structure that contains record parameters.

Return values

Return value	Description
SUCCESS	
ADIERR_INVALID_CALL_STATE	Function not valid in the current call state.
CTAERR_BAD_ARGUMENT	Invalid ***encoding*** or NULL ***buffer***.
CTAERR_FUNCTION_ACTIVE	Function already started.
CTAERR_INVALID_CTAHD	Context handle is invalid.
CTAERR_INVALID_STATE	Function not valid in the current port state.
CTAERR_OUTPUT_ACTIVE	Record failed because there is another active output function.
CTAERR_RESOURCE_CONFLICT	Silence detector is in use by **adiStartEnergyDetector**.
CTAERR_SVR_COMM	Play or record communication error.

Events

The ADIEVN_RECORDING_DONE event is generated when the recording operation terminates. The event size field contains the total number of bytes written to the buffer. The value field contains one of the following termination reasons or error codes:

Reason	Description
CTA_REASON_FINISHED	Buffer is filled.
CTA_REASON_MAXTIME_AUDIO	Maximum recording time reached. Applicable for pass-through audio encoding types only.
CTA_REASON_MAXTIME_VIDEO	Maximum recording time reached. Applicable for pass-through video encoding types only.
CTA_REASON_NATIVE_COMPANION_RECORD_STOPPED	Companion pass-through record channel stopped. For example, there is a pass-through audio channel and a pass-through video channel that are companion channels because they have matching ***video_ephd*** and ***audio_ephd*** input parameters. If one of the two channels stops recording for any reason, the companion pass-through channel is also stopped with this reason code.
CTA_REASON_NO_VIDEO	No video detected.
CTA_REASON_NO_VOICE	No voice detected.
CTA_REASON_RELEASED	Call terminated.
CTA_REASON_STOPPED	Stopped by application request.
CTA_REASON_VIDEO_END	Video packets not detected for specified timeout period. Applicable for pass-through video encoding types only.
CTA_REASON_VOICE_END	User stopped speaking.
CTAERR_FUNCTION_NOT_AVAIL	Required DSP file not loaded on the board.
CTAERR_xxx or ADIERR_xxx	Record failed.

See also

adiPlayMMFromMemory

Example

```
if ((ret = SetRecordParms(VideoCtx[i].ctahd, &recParms)) != SUCCESS)
{
    printf("Failed to setup Record Parameters\n");
    exit(1);
}

ret = adiRecordMMToMemory(VideoCtx[i].ctahd,
mmParm.ADIvidEncoder,       // Encoding
VideoCtx[i].data_buffer,    // Buffer to receive recorded video
mmParm.video_buffer_size,   // Buffer size
videoEP_filterhd,           // Video EndPoint Filter Handle
audioEP_filterhd,           // Audio EndPoint Filter Handle
&recParms);                 // Record parameters

if (ret != SUCCESS)
{
    printf("Failed to Record to memory, %x\n", ret);
    exit(1);
}
```

ADI_MM_RECORD_PARMS

Contains parameters for recording multimedia data. This structure is used by **adiRecordMMAsync** and **adiRecordMMToMemory**.

Definition

The ADI_RECORD_MM_PARMS structure contains fields that are specific to video recording and contains fields that are not applicable to video recording. In the structure definition shown below:

- Lines in bold are specific to video recording
- Crossed out fields are not applicable to video recording

```
typedef struct
{
    DWORD size;                /* Size of this structure                  */
    DWORD DTMFabort;           /* Abort on DTM                            */
~~    INT32 gain;                /* Recording gain in dB                    */~~
~~    /* [SLC parms (used if silence det)] */~~
    DWORD novoicetime;         /* Length of initial silence to stop       */
                               /* Recording (ms); use 0 to                */
                               /* deactivate Initial silence              */
                               /* detection.                              */
    DWORD silencetime;         /* Length of silence to stop               */
                               /* recording After voice has been          */
                               /* detected(ms);Use 0 to deactivate        */
    INT32 silenceampl;         /*Qualif level for silence (dBm)           */
                               /* The maximum voice signal level that     */
                               /* is considered to be silence.            */
    WORD silencedeglitch;      /* Deglitch while qualifying(ms)           */
                               /* The maximum non-silent interval         */
                               /* that is ignored by the silence          */
                               /* detector. Any sounds that last          */
                               /* longer than this value reset            */
                               /* the silence detector                    */
~~    DWORD beepfreq;            /* beep frequency(Hz)                      */~~
~~    INT32 beepampl;            /* beep amplitude(dBm)                     */~~
~~    DWORD beeptime;            /* Beep time(ms) 0=no beep                 */~~
~~    WORD  AGCEnable;           /* Enable AGC; use 1 to activate           */~~
~~    INT32 AGCtargetampl;       /* target AGC level(dBm)                   */~~
~~    INT32 AGCsilenceampl;      /* Silence level(dBm)                      */~~
~~    DWORD AGCattacktime;       /*Attack time(ms)                          */~~
~~    DWORD AGCdecaytime;        /* Decay time(ms)                          */~~
    BOOL startonvideoIframe;   /*1=start collection on                    */
                               /*intial I-frame                           */
                               /*0=start immediately                      */
                               /*Native video channels only.              */
    DWORD maxtime;             /*length of time(100 ms increments)after   */
                               /*which recording will terminate. Set to   */
                               /* 0 to deactivate.  Native audio          */
                               /* and video channels                      */
    DWORD novideotime;         /*for startonvideoIframe=0,length          */
                               /*of time (ms) to stop recording           */
                               /* for no receipt of video.                */
                               /*for startonvideoIframe=1, length         */
                               /*of time to stop recording for no         */
                               /*receipt of I-frame.                      */
                               /*0=deactivate.Native video                */
                               /*channels only.                           */
    DWORD videotimeout;        /* length of time(ms)without               */
                               /* receiving video to stop video           */
                               /* recording after video has been          */
                               /* detected. Set to 0 to deactivate        */
                               /* Native video channels only.             */
} ADI_MM_RECORD_PARMS;
```

Audio pass-through or transcoding channel parameters

For an audio pass-through or transcoding channel, the following parameters in the ADI_MM_RECORD_PARMS structure apply:

- novoicetime
- silencetime
- maxtime
- silenceampl
- silencedeglitch

If audio silence detection is enabled, the DTMFabort parameter also applies to an audio channel. However, in the context of 3G-324M communications, NMS recommends that you set this parameter to 0 and avoid in-band DTMF detection. Those signals are typically carried out-of-band in H.245 User Input Indication messages.

Video pass-through channel parameters

For a video pass-through channel, the following parameters in the ADI_MM_RECORD_PARMS structure apply:

- startonvideoIframe
- maxtime
- novideotime
- videotimeout

The same parameter structure is used for both audio and video channels. Only the parameters that apply for the particular audio or video channel are considered by the NMS record component.

The following table describes the video recording fields in the ADI_MM_RECORD_PARMS structure. For all fields, specifying NULL designates default values.

Field	Description
startonvideoIframe	The application has the option to begin recording on both the audio and video channels. Recording can begin immediately or when the first video I-frame is detected on the video channel with the startonvideoIframe parameter in **adiRecordMMToMemory** or **adiRecordMMAsync**. When startonvideoIframe = 0, audio and video recording begins immediately. When startonvideoIframe = 1, the audio and video filters begin recording as follows: • On the video channel, the filter discards video packets until detection of the first video I-frame. • When the first I-frame is detected, the filter begins collection of video packets. • The filter issues a message to the voice manager task on the board to begin audio recording on the companion audio channel, if necessary. The application is not required to disable the RTP endpoint of the audio channel. The voice manager task on the board waits until the first video I-frame is detected and then creates the companion audio channel, preventing audio reception prior to the first video I-frame. Audio and video streaming to the host starts when the video I-frame is detected. A companion audio channel is a channel that was started with the same endpoint handles as for the video record. The video record must be started first.
maxtime	When applied to a video pass-through channel, and if not set to 0, this maximum record duration timer gets started on a different event, depending on the startonvideoIframe parameter: • If startonvideoIframe = 1, it starts on video I-frame detection. • If startonvideoIframe = 0, it starts on the first video packet received.
novideotime	Indicates the amount of time the ADI service waits before stopping the recording process if it does not receive video. A value of 0 (zero) de-activates this field. The effect of this field (when non-zero) depends upon the value of the startonvideoIframe field: • If startonvideoIframe = 0, then ADI waits this amount of time before stopping the recording process, if no video is received. • If startonvideoIframe = 1, then ADI waits this amount of time before it stops the recording process if no video I-frame is received.
videotimeout	Indicates the amount of time the ADI service waits before stopping the recording process, if the video stream stops. The starting point for the time measurement is the initial receipt of video. A value of 0 (zero) de-activates this feature.

For information on the other fields in the ADI_MM_RECORD_PARMS structure, see the *ADI Service Developer's Reference Manual*.

6 MSPP service

Video enhancements to the MSPP service

The MSPP service creates and controls audio and video RTP endpoints. The Video Messaging Server Interface enhances the Natural Access MSPP service to support video endpoints. **mspGetFilterHandle** translates the MSPP endpoint handle acquired from **mspCreateEndpoint** into the CG board filter endpoint handle, which is a required parameter for the ADI service multimedia play and record functions. The MSPP endpoint handle is required for **adiPlayMMFromMemory**, **adiPlayMMAsync**, **adiRecordMMToMemory**, and **adiRecordMMAsync**. For more information about the MSPP service and **mspGetFilterHandle**, see the *MSPP Service Developer's Reference Manual*.

The Video Messaging Server Interface also supports the use of RTCP for communicating skew information to the host application and IP destination through RTP endpoints. For information, see *Configuring endpoints to use audio/video synchronization* on page 36.

Creating endpoints

The application must create audio and video MSPP endpoints. Each of these endpoints has an identifier that must be used when creating the endpoint. These identifiers are used in the eEpType field in the MSP_ENDPOINT_ADDR structure and in the eParmType field in the MSP_ENDPOINT_PARMS structure. The following table lists the identifiers for endpoints in the Video Messaging Server Interface:

Endpoint	Identifier
Video RTP	MSP_ENDPOINT_RTPFDX_VIDEO MSP_ENDPOINT_RTPFDX_VIDEO_V6
Audio RTP	MSP_ENDPOINT_RTPFDX MSP_ENDPOINT_RTPFDX_V6

The following code sample shows how to declare that an endpoint is a server video RTP endpoint:

```
VideoCtx[i].rtpEp.Addr.eEpType = MSP_ENDPOINT_RTPFDX_VIDEO;
VideoCtx[i].rtpEp.Param.eParmType = MSP_ENDPOINT_RTPFDX_VIDEO;
```

All endpoints are created using **mspCreateEndpoint**.

Note: The IPv4 or IPv6 type of video endpoint is used on the Messaging Server Interface regardless of the video format being used. The endpoint can handle MPEG-4, H.263, and H.264 video.

For more information about creating endpoints and the MSP_ENDPOINT_ADDR structure, see the *MSPP Service Developer's Reference Manual*. For more information about the MSP_ENDPOINT_PARMS structure, see *MSP_ENDPOINT_PARMS* on page 74.

Creating server pass-through channels

The same channel, RTPPassthruSimplex, is used for both audio bridging configurations and video bridging configurations.

A pass-through channel is created and connected in the same way a regular Fusion RTP switching simplex channel is created and connected. However, a pass-through channel uses the identifier RTPPassthruSimplex instead of RTPSwitchingSimplex.

For more information about RTP switching channels, see the *Fusion Developer's Manual*. For more information about audio and video bridging configurations, see *Defining a media bridging configuration* on page 32.

Video endpoint MSPP commands and queries

The following commands and queries can be sent to the video RTP endpoint:

Commands and queries (msp command structure)	**Description**
MSP_CMD_RTPFDX_CONFIG (msp_ENDPOINT_RTPFDX_CONFIG)	Specifies configuration parameters for the endpoint. For more information, see the *MSPP Service Developer's Reference Manual.*
MSP_CMD_RTPFDX_MAP (msp_ENDPOINT_RTPFDX_MAP)	Assigns a payload ID to a vocoder. For more information, see the *MSPP Service Developer's Reference Manual.*
MSP_CMD_RTPFDX_LINK_EVENTS (msp_ENDPOINT_RTPFDX_LINK_EVENTS)	Enables or disables reporting of link availability transitions through unsolicited events. For more information, see the *MSPP Service Developer's Reference Manual.*
MSP_CMD_RTPFDX_STATUS (msp_ENDPOINT_RTPFDX_STATUS)	Returns information about the RTP endpoint filter state. For more information, see the *MSPP Service Developer's Reference Manual.*
MSP_CMD_RTPFDX_RTCP_EVENTS (msp_ENDPOINT_RTPFDX_RTCP_EVENTS)	Enables or disables reporting of RTCP events through unsolicited events. For more information, see the *MSPP Service Developer's Reference Manual.*
MSP_CMD_RTPFDX_CALC_SKEW_OFFSET (msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC)	Enables the calculation of skew offsets used for audio/video synchronization purposes for the specified full-duplex or simplex receive audio and video endpoints. For more information, see *Configuring endpoints to use audio/video synchronization* on page 36.
MSP_CMD_RTPFDX_VIDEO_SKEW_TIME (msp_ENDPOINT_RTPFDX_SET_VID_SKEW)	Sets a video skew value to send to the IP destination. The skew value can be used for audio/video synchronization purposes for the specified full-duplex video endpoint. For more information, see *Enabling RTP endpoints to send video skew values to the IP destination* on page 39.

Creating and sending MSPP commands

You can create an MSPP command by using the mspBuildCommand macro along with an MSPP endpoint command. You can send an MSPP command to the specified endpoint by using **mspSendCommand**.

The mspBuildCommand macro and **mspSendCommand** are described below. The MSPP endpoint commands are described Video endpoint MSPP commands and queries.

The mspBuildCommand macro

The mspBuildCommand macro builds a command by concatenating the endpoint filter ID with an endpoint command. It is defined as follows:

```
#define mspBuildCommand(filter,command) ((filter << 8) | command)
```

For more information about the mspBuildCommand macro, see the *MSPP Service Developer's Reference Manual*.

The mspSendCommand function

mspSendCommand sends a concatenated command to an MSPP endpoint. It is defined as follows:

```
DWORD mspSendCommand(MSPHD msphd, DWORD command, void *buffer, DWORD size)
```

Where:

- ***msphd*** is a unique MSPP endpoint handle (obtained when creating the endpoint with **mspCreateEndpoint**).
- ***command*** is a valid command produced by using the mspBuildCommand macro.
- ****buffer*** is a pointer to a structure that contains the value to assign to the command. The value of this argument is NULL, if there is no associated structure.
- ***size*** is the size of the structure. This value of this argument is 0, if there is no associated structure.

For more information about **mspSendCommand**, see the *MSPP Service Developer's Reference Manual*.

Creating and sending MSPP queries

You can create an MSPP query by using the mspBuildQuery macro along with an MSPP endpoint command. You can send an MSPP query to the specified endpoint by using **mspSendQuery**.

The mspBuildQuery macro and **mspSendQuery** are described below. The video-enhanced MSPP endpoint commands and queries are described in *Video endpoint MSPP commands and queries* on page 71.

The mspBuildQuery macro

The mspBuildQuery macro builds a query by concatenating the endpoint filter ID with a query. It is defined as follows:

```
#define mspBuildQuery(filterid,queryid)    ((filterid << 8) | query)
```

For more information about the mspBuildQuery macro, see the *MSPP Service Developer's Reference Manual*.

The mspSendQuery command

mspSendQuery sends a concatenated query to an MSPP endpoint. It is defined as follows:

```
DWORD mspSendQuery(MSPHD msphd, DWORD query)
```

Where:

- ***msphd*** is a unique MSPP endpoint handle (obtained when creating the endpoint with **mspCreateEndpoint**).
- ***query*** is a valid query produced by using the mspBuildQuery macro.

For more information about **mspSendQuery**, see the *MSPP Service Developer's Reference Manual*.

MSPP video-enhanced structures

The following video-enhanced MSPP structures work with video RTP endpoints in the Video Messaging Server:

- MSP_ENDPOINT_PARMS
- msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC
- msp_ENDPOINT_RTPFDX_SET_VID_SKEW
- RTPRTCP_ENDPOINT_PARMS
- RTPRTCP_V6_ENDPOINT_PARMS

MSP_ENDPOINT_PARMS

Use this structure to set configuration parameters for audio and video endpoints. The MSP_ENDPOINT_PARMS structure is used by **mspCreateEndpoint**.

The parameters that you specify in the MSP_ENDPOINT_PARMS structure depend on the type of endpoint you create. For example, if you create an MPEG-4 or H.263 video endpoint for IPv4, use the RTPRTCP_ENDPOINT_PARMS structure as a parameter for MSP_ENDPOINT_PARMS. If you create a video endpoint for IPv6, use the RTPRTCP_V6_ENDPOINT_PARMS structure as a parameter for MSP_ENDPOINT_PARMS.

```
typedef struct tag_MSP_ENDPOINT_PARMS
{
    DWORD   size;          // size of MSP_ENDPOINT_PARAMS)
    DWORD   eParmType;     // MSP_ENDPOINT_DS0, MSP_ENDPOINT_RTPFDX, etc

union
    {
        DS0_ENDPOINT_PARMS          DS0;
        PKTMEDIA_ENDPOINT_PARMS     Pktmedia;
        MONITOR_ENPOINT_PARMS       Monitor;
        RTPRTCP_ENDPOINT_PARMS      RtpRtcp;
        UDP_ENDPOINT_PARMS          Udp;
        T38UDP_ENDPOINT_PARMS       T38Udp;
        TPKT_ENDPOINT_PARMS         Tpkt;
        MUX_ENDPOINT_PARMS          Mux;
        UNDEFINED_ENDPOINT_PARMS    Undefined;
        // Structure may be expanded to define new Endpoints
        RTPRTCP_V6_ENDPOINT_PARMS   RtpRtcpV6
        UPD_V6_ENDPOINT_PARMS       UdpV6
    } EP;

} MSP_ENDPOINT_PARAMETER;
```

For more information about **mspCreateEndpoint**, see the *MSPP Service Developer's Reference Manual.*

msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC

Use this structure to enable or disable the calculation of skew values based on incoming RTCP sender reports and RTP packets for full-duplex and simplex receive endpoints. This structure is used for both audio and video endpoints.

This structure is used with the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command, which is described in *Configuring endpoints to use audio/video synchronization* on page 36.

Definition

```
typedef struct {
U32   enable;  // Set 4th bit  to:  0=disable;  1=enable
} msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC;
```

Field

Field	Default	Description
enable	0	Controls skew value calculations. Valid values are: • 0 in 4th bit - disable skew calculation. • 1 in 4th bit - enable skew calculation. Use the RTCP_ENABLE_RCV_SKEW_CALC macro to set the appropriate bit. For more information, see *MSP_ENDPOINT_PARMS* on page 74.

msp_ENDPOINT_RTPFDX_SET_VID_SKEW

Use this video-specific structure to set a video skew value to be signaled in RTCP sender report packets for full-duplex and simplex send video RTP endpoints.

This structure is used with the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command, which is described in *Enabling RTP endpoints to send video skew values to the IP destination* on page 39.

Definition

```
typedef struct {
U32   vidSkew;  // Set value to 0-4095ms in 5th-16th bits
} msp_ENDPOINT_RTPFDX_SET_VID_SKEW;
```

Fields

Field	Default	Description
vidSkew	0	Video skew value in ms, using the 5th-16th bits of the DWORD. Valid values are 0 to 4095 ms. Use the RTCP_VIDEO_SKEW macro to set the appropriate skew value. If the skew value indicates video lead time, as opposed to video lag time, use the RTCP_VIDEO_LEADS_AUDIO macro as well. For more information about these macros, see *RTPRTCP_ENDPOINT_PARMS* on page 76. For more information about audio/video synchronization, see *Configuring endpoints to use audio/video synchronization* on page 36.

RTPRTCP_ENDPOINT_PARMS

Use this structure to set configuration parameters for an RTP IPv4 endpoint. This structure is used as a parameter for the MSP_ENDPOINT_PARMS structure, which is used with **mspCreateEndpoint**.

```
typedef struct tag_RTPRTCP_ENDPOINT_PARMS
{

    DWORD size;
    // QoS parameters
    BYTE TypeOfService;     // Default = 0, type of service in IP header
    DWORD startRtcp;        // Set this to non-zero to start RTCP
                            // session. RTCP_SESSION_PARMS structure must be
                            // filled in for the RTCP session
    /* RTCP parameters */
    RTCP_SESSION_PARMS rtcpParms;

    DWORD RtpTsFreq;              // Default=8000, timestamp frequency
                                  // For a Video Endpoint, RtpTsFreq is fixed to 90000
                                  // and cannot be modified
    DWORD Session_bw;             // Default=64000, session bandwidth
    DWORD dtmf_event_control;     // Control DTMF RTP Event generation
    DWORD frameQuota;             // RTP Assembly frame quota
                                  // For a Video Endpoint, frameQuota is fixed to 1
                                  // and cannot be modified
    DWORD linkEvents;             // Controls link events
    RTP_PAYLOAD_MAP  PayloadMap;

} RTPRTCP_ENDPOINT_PARMS;
```

Note: For more information on the RTCP_SESSION_PARMS structure, refer to the *MSPP Service Developer's Reference Manual*.

When using RTCP for audio/video synchronization purposes, the startRtcp parameter is used as a bit field containing several definitions. The following macros can be used to set the bits in the startRtcp field:

```
#define  RTCP_ENABLE(x)                (x)=1                        //Sets value to 1
#define  RTCP_SET_0_INTERVAL(x)       ((x) | 2)                     //Sets 2nd bit
#define  RTCP_VIDEO_LEADS_AUDIO(x)    ((x) | 4)                     //Sets 3rd bit
#define  RTCP_ENABLE_RCV_SKEW_CALC(x) ((x) | 8)                     //Sets 4th bit
#define  RTCP_VIDEO_SKEW(x,y)         ((x) | (((y) & 0x0fff) << 4))
   //Sets 5th to 16th bits to value of y (max of 4095)
```

The following table describes how to use these macros to set bits in the startRtcp field:

Use this macro...	**To set this bit...**	**Description**
RTCP_ENABLE	0	Enables RTCP for an endpoint. Use this macro for both endpoints of the audio/video stream pair to be synchronized.
RTCP_SET_0_INTERVAL	1	Determines how quickly the RTCP Sender Report is sent: • When set to 1, the first RTCP Sender Report is generated and transmitted between 0 and 1 second after the first RTP packet is transmitted for the stream. • When not set (macro is not used), the first RTCP Sender Report is sent five seconds after the first RTP packet is transmitted for the stream. Use this macro for both endpoints of the audio/video stream pair to be synchronized. NMS recommends that you always use this macro.
RTCP_VIDEO_LEADS_AUDIO	2	(Video endpoints only) Signals to the video endpoint whether the video data stream leads or lags the audio data stream. Set this bit to 1, if it is known that the skew between transmitted video and transmitted audio for a synchronization stream pair is such that video leads audio. Otherwise, do not use this macro.
RTCP_ENABLE_RCV_SKEW_CALC	3	Calculates an offset value that can be sent to the application as an unsolicited event, MSPEVN_SKEW_OFFSET. Use this macro for both endpoints of the audio/video stream pair to be synchronized. To determine the skew between audio and video, compare the returned offset values of the endpoints of the stream pair.
RTCP_VIDEO_SKEW	4 - 15	Sets a value between 0 and 4095 ms of video skew that will be communicated to the IP destination by RTCP Sender Reports.

If the application enables RTCP reports through the startRtcp parameter in the RTPRTCP_ENDPOINT_PARMS structure, it must also include the following substructure:

```
typedef struct RTCP_RTCP_SESSION_PARMS
{
DWORD       forwardPkts;
char        cname[32];
char        name[32];
char        email[32];
char        phone[32];
char        location[32];
char        tool[32];
char        note[32];
} RTCP_SESSION_PARMS;
```

For more information about:

- Using RTCP for audio/video synchronization purposes, see *Configuring endpoints to use audio/video synchronization* on page 36.
- Creating endpoints, see *Creating endpoints* on page 69.
- **mspCreateEndpoint**, see the *MSPP Service Developer's Reference Manual.*

RTPRTCP_V6_ENDPOINT_PARMS

Use this structure to set configuration parameters for an RTP IPv6 endpoint. This structure is used as a parameter for the MSP_ENDPOINT_PARMS structure, which is used with **mspCreateEndpoint**.

```
typedef struct tag_RTPRTCP_V6_ENDPOINT_PARMS
{
    DWORD size;
    BYTE  trafficClass;  // Indicates the class or priority of the IPv6 packet
    DWORD flowLabel;     // Indicates the specific sequence the IPv6 packet belongs to
    DWORD startRtcp;               // Set this to non-zero to start RTCP
                                   //  session. RTCP_SESSION_PARMS structure must be
                                   //  filled in for the RTCP session
    /* RTCP parameters */
    RTCP_SESSION_PARMS rtcpParms;
    DWORD RtpTsFreq;               // Default=8000, timestamp frequency
    DWORD Session_bw;              // Default=64000, session bandwidth
    DWORD dtmf_event_control;      // Control DTMF RTP Event generation
    DWORD frameQuota;              // RTP Assembly frame quota
    DWORD linkEvents;              // Controls link events
    RTP_PAYLOAD_MAP  PayloadMap;
} RTPRTCP_V6_ENDPOINT_PARMS;
```

When using RTCP for audio/video synchronization purposes, the startRtcp parameter is used as a bit field containing several definitions.

For more information about:

- Using the startRtcp parameter, see *RTPRTCP_ENDPOINT_PARMS* on page 76.
- Creating endpoints, see *Creating endpoints* on page 69.
- **mspCreateEndpoint**, see the *MSPP Service Developer's Reference Manual.*

Timestamp smoothing

To prevent RTP timestamp interruptions when switching from one source to another during a call, a timestamp smoothing (TS) algorithm is implemented in the server RTP audio and video endpoints. Examples of changing sources include:

- Switching from a bridging configuration to a playback configuration.
- Performing consecutive playbacks from different files as part of the same call.

The transmitting RTP endpoint uses a real-time clock to correlate the time difference between consecutive arriving packets and the difference in their timestamps. Only one input channel can be enabled at one time. A threshold parameter called TS Threshold (TST) determines if there is an interruption between sequential packets.

If | (TimeStamp1 – TimeStamp2) – (PktArrivalTime1 – PktArrivalTime2) | > TST, for packets arriving into the transmitting RTP endpoint, the endpoint generates a timestamp for Packet 2. This timestamp is based on the difference between the arrival times on packets 1 and 2, and not on the difference in timestamps between input TimeStamp1 and TimeStamp2.

For example, if the endpoint is transmitting single frames of AMR packets, then normally, if the threshold is not exceeded, TimeStamp2 = TimeStamp1+20ms. However, if the threshold is exceeded, the timestamp smoothing algorithm sets TimeStamp2 = TimeStamp1 + (PktArrivalTime2 - PktArrivalTime1).

Pass-through channels and IVR/Fusion channels supply packets with timestamps that are correlated to their measured arrival times because they are throttled within a single real-time system on the CG board. If these two paths are the only source of the packets, the threshold can be set to a small time value, for example 30 ms. the packets arriving through the pass-through channels, however, are subject to network jitter. The jitter can cause the timestamp smoothing algorithm to be invoked if the threshold is set too low. It is better to avoid this and, as such, the timestamp smoothing threshold parameter is application-configurable for both audio and video RTP endpoints. Each endpoint can be properly tuned to network conditions so it does not confuse the packets arriving late and the packets arriving from a new source (through a new channel).

Set the threshold parameter TST with **mspSendCommand** using a command identifier of MSP_CMD_RTPFDX_SET_TS_SMOOTHING_THRESH. The resolution of the threshold value is given in ms, and the value must lie in the 100 ms to 2000 ms range. The default threshold value is 100 ms. The following code fragment illustrates how to set the threshold:

```
msp_ENDPOINT_RTPFDX_SET_TS_THRESH thresh_cmd;
DWORD  command;

thresh_cmd.value = H2NMS_DWORD(150);  //150 ms, Endian adjusted


command = mspBuildCommand(MSP_ENDPOINT_RTPFDX_AUDIO,
MSP_CMD_RTPFDX_SET_TS_SMOOTHING_THRESH);

ret = mspSendCommand(EP_handle, command, (void*) thresh_cmd,
sizeof(msp_ENDPOINT_RTPFDX_SET_TS_THRESH));
```

7 Multimedia File Interface library (3GP)

Multimedia File Interface library overview

The Multimedia File Interface (MMFI) library enables an application to merge audio and video media streams into a 3GP format file, and inversely, to split a 3GP format file into two separate media streams.

Using the MMFI library, a Video Access application can:

- Merge an NMS packetized format video stream and an NMS packetized format audio stream into a 3GP file, formatted in 3GPP Basic Profile or 3GPP Streaming Server Profile.
- Merge an NMS packetized format video stream and a raw AMR-NB IF2 audio stream into a 3GP file, formatted in 3GPP Basic Profile.
- Split a 3GP file into its constituent media streams, provided those streams are in the following format:
    - NMS packetized H.263 baseline bit video stream
    - NMS packetized H.263+ profile 3 bit video stream
    - NMS packetized H.264 baseline profile bit video stream
    - NMS packetized MPEG-4 simple profile bit video stream
    - NMS packetized AMR-NB IF2 bit audio stream
    - Raw AMR-NB IF2 bit audio stream
- Obtain format information from an existing 3GP file, such as contents and stream size.
- Create a 3GP file with a synchronization table that contains one synchronization point for every I-frame.
- Randomly access specific points in time within a file using the 3GP file's synchronization table.
- Create a 3GP file with hint tracks that contain instructions for packaging the audio and video tracks into a channel.
- Use a 3GP file's hint tracks to format audio and video media tracks according to the hint track information.
- Create a 3GP file with audio and video tracks synchronized using skew correction.
- Store SDP data to a 3GP file. Retrieve SDP data from a 3GP file.

Video Access merges audio and video streams into a 3GP file by formatting and storing the audio and video buffers resulting from **adiRecordMMToMemory** or **adiRecordMMAsync** calls. Video Access splits a file into audio and video streams by extracting the audio and video media tracks from a 3GP file and playing them with the **adiPlayMMFromMemory** or **adiPlayMMAsync** calls.

The MMFI library does not perform media transcoding during either merge or split operations.

Input and output formats

There are three types of input or output formats used by the MMFI library to merge streams or split files:

- NMS packetized steam format (audio stream, video stream, or both)

  Note: In the MMFI library header files, NMS packetized formats are referred to as native formats.

- Raw AMR-NB IF2 stream format (audio stream only)
- 3GP file format (including an audio stream, a video stream, or both)

The following table shows how the types of streams are utilized in a 3GP file.

This audio stream format	Plus this video stream format	Can be in this 3GP File Format
NMS packetized	NMS packetized	3GPP Basic Profile or Streaming Server Profile
Raw AMR-NB IF2	NMS packetized	3GPP Basic Profile

NMS packetized format

The NMS packetized format can be one of the following:

- H.263 baseline profile 0 or H.263+ profile 3, level 10 - 30 (QCIF, CIF) video elementary bit stream.

  Video bit streams must conform to ITU-T Recommendation H.263 Annex X as defined in ITU-T Recommendation H.263 1998 and 2000 and 3GPP specifications TS.26.111 and TS.26.911.

- H.264 baseline profile level 1 – 1.2 (QCIF, CIF) video elementary bit stream formatted in the NMS-packetized proprietary format.

  Video bit streams must conform to ITU-T Recommendation and H.264 and 3GPP specifications TS.26.111 and TS.26.911.

  Note: For the H.264 video codec, raw video to 3GP format conversion is not supported.

- MPEG-4 simple profile level 0 - 3 (QCIF, CIF) video elementary bit stream formatted in the NMS packetized proprietary format.

  The library expects an ISO/IEC 14496 simple profile level 0 - 3 bit stream. Time stamping and headers (in particular VOS/VO/VOL headers) are expected to be compliant with ISO/IEC 14496 and 3GPP TS 26.110 /TS 26.111. This information is needed in the correct format to allow the computation of parameters for the 3GP file formatting.

- AMR-NB IF2 elementary audio bit stream formatted in the NMS packetized proprietary format.

  Audio bit streams must conform to ETSI/AMR, Adaptive Multi-Rate speech codec (AMR), in IF2 framing format, in any of the eight AMR compressed data rates.

The NMS packetized formats are supported by **adiRecordMMToMemory**, **adiRecordMMAsync**, **adiPlayMMFromMemory**, and **adiPlayMMAsync**.

Raw AMR-NB IF2 format

This format consists of a raw AMR-NB IF2 elementary audio bit stream, as supported by **adiRecordMMToMemory**, **adiRecordMMAsync**, **adiPlayMMFromMemory**, and **adiPlayMMAsync** when using the audio transcoding configuration.

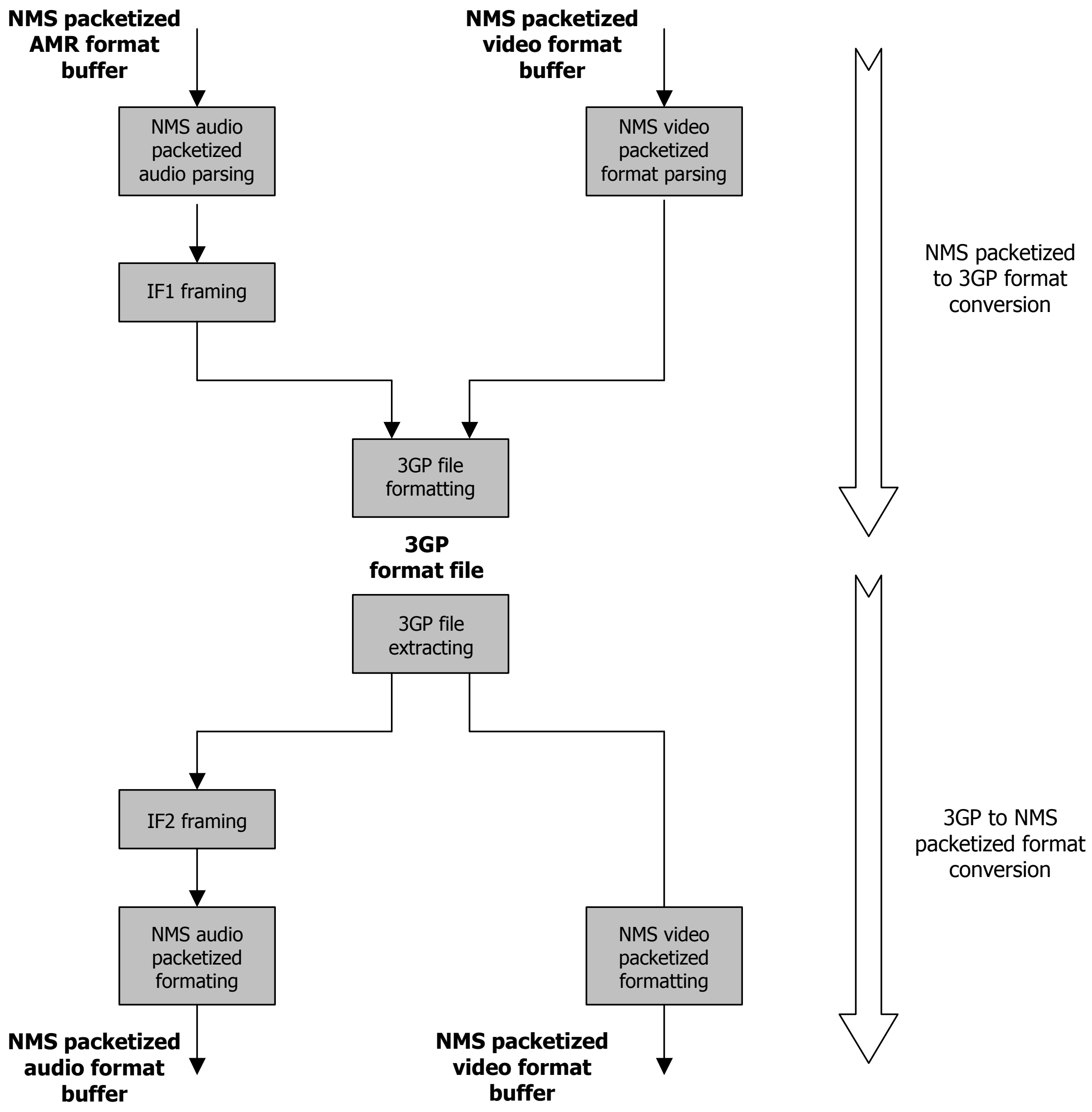
3GP format

The 3GP file format is compliant with the 3GPP TS 26.244 specification and conforms to 3GP file format Basic Profile or 3GPP Streaming Server Profile.

The 3GP file contains one video track (either H.263, H.264, or MPEG-4), one audio track (IF1 AMR NB audio elementary bit stream), or both. It can also contain hint tracks for one or both of the audio and video streams.

3GP merge and split processes

The 3GP merge and split processes are shown in the following illustration using NMS packetized format audio and video streams:


NMS packetized AMR format buffer
NMS packetized video format buffer
NMS audio packetized audio parsing
NMS video packetized format parsing
IF1 framing
NMS packetized to 3GP format conversion
3GP file formatting
3GP format file
3GP file extracting
IF2 framing
3GP to NMS packetized format conversion
NMS audio packetized formating
NMS video packetized formatting
NMS packetized audio format buffer
NMS packetized video format buffer

There are two types of format conversions:

- NMS packetized to 3GP
- 3GP to NMS packetized

NMS packetized to 3GP format conversion

The following table shows how the MMFI converts data from NMS-packetized to 3GP format:

Step	Description
1	Converts video NMS packetized format data into a raw video bit stream. Basic checking on NMS headers can optionally be performed.
2	Converts audio NMS packetized format data into an AMR audio bit stream (AMR-NB IF2 format). Basic checking on NMS headers can optionally be performed. **Note:** The MMFI skips this step if the audio data is in raw format instead of NMS packetized format.
3	Converts the IF2 AMR bit stream into IF1 AMR bit stream.
4	Generates a *.3gp* file with 3GP meta data, the video elementary bit stream (MPEG-4, H.263, or H.264) and the IF1 AMR NB audio elementary bit stream. The 3GP formatting function does not perform any audio or video bit stream adaptation or transcoding. It only merges audio and video bit streams input into 3GP file format compliant with 3GPP TS 26.244.

The compressed media in a 3GP output file can be optionally limited in size and in duration. The maximum size is specified in bytes, and applies to the total size of the 3GP file or equivalent stream. The maximum time is specified in milliseconds.

3GP to NMS packetized format conversion

The following table shows how the Video Messaging Server Interface converts data from 3GP format to NMS packetized format:

Step	Description
1	Extracts from the *.3gp* file the video elementary bit stream (H.263, H.264, or MPEG-4) and the IF1 AMR NB audio elementary bit stream.
2	Converts the IF1 AMR bit stream into IF2 AMR bit stream.
3	Converts the IF2 AMR audio bit stream into AMR audio data in NMS packetized format. **Note:** The MMFI skips this step if the audio data is to be converted to raw format instead of NMS packetized format.
4	Converts the video bit stream into video data in NMS packetized format.

Multimedia File Interface features

The MMFI library supports the following 3GP features:

Feature	**Description**
Random access	Allows a user to seek to a specific time within a video track, jumping ahead or back in a 3GP file during a play operation.
Skew correction	Allows a user to synchronize an audio and video stream during a 3GP file record operation by specifying a skew correction value for the multimedia file.
Hint tracks	Allows a user to specify that hint tracks be created during a 3GP file record operation. Hint tracks allow a streaming server to create RTP streams from a 3GP file without knowledge of the media type, compression, or payload format.
SDP	Allows an application to store SDP data to a 3GP file and retrieve SDP data from a 3GP file.

Random access

The random access feature allows a user to seek to a specific time within a video track, jumping ahead or back in a 3GP file during a play operation.

Sync points table

A 3GP file must have a sync point table in order to support random access. A sync point table is created when the video stream is opened for recording, if specified by the application. A sync point table is then filled in as the video stream is recorded. There is one entry in the sync point table for every I-frame written to the 3GP file. Each entry in the sync point table is a video sample number to which a 3GP file can be randomly accessed.

Seek operations

The operation of seeking within a 3GP file involves aligning the file's track position pointer with a sync point. When moved, the track position pointer is updated for all tracks in the 3GP file. An application may seek to a sync point by calling one of the following functions:

Function	**Description**
mmSeekToTime	Allows the application to seek to a specific time specified in milliseconds within a 3GP file, relative to the beginning of the file.
mmSeekToNextSyncPoint	Allows the application to seek to the next sync point after the current track position pointer.
mmSeekToPrevSyncPoint	Allows the application to seek to the previous sync point before the current track position pointer.

See *Recording to a 3GP file* on page 88 for an example of when to specify the creation of a sync point table in the video stream.

See *Random access call flow* on page 94 for a call flow example that uses seek functions to do random access.

Skew correction

The skew correction feature allows a user to synchronize an audio and video stream during a 3GP file record operation by specifying a skew correction value for the multimedia file. The value of the skew correction determines how synchronization is achieved.

Depending on the value of the skew correction, the audio stream's record operation works as follows:

- If the audio stream lags the video stream, the skew correction is specified (in milliseconds) as a positive value.
- If the audio stream leads the video stream, the skew correction is specified (in milliseconds) as a negative value.

See *Recording to a 3GP file* on page 88 for call flow examples.

Hint tracks

The hint tracks feature allows an application to request that hint tracks be created during a video or audio stream record operation. A hint track is a track in the 3GP file that contains hint information for an audio or video media track. Hint tracks allow a streaming server to create RTP streams from a 3GP file without knowledge of the media type, compression, or payload format.

To create hint tracks in the 3GP file, the user sets a flag when opening the audio and video streams. As the audio and video streams are written to the multimedia file, the hint tracks are automatically built.

The hint tracks feature improves performance by eliminating the overhead incurred by low level RTP packetization processing when playing a multimedia file. During a play operation, the hinting information is used to properly packetize the track’s media frames into NMS native packets.

Note: Hint tracks can be created from MS packetized media formats only.

See *Recording to a 3GP file* on page 88 for an example of specifying the use of hint tracks in the audio and video streams.

SDP

The SDP feature allows an application to store SDP data to a 3GP file and retrieve SDP data from a 3GP file. The application uses **mmSetSDPinfo** to store SDP information and uses **mmGetSDPinfo** to retrieve SDP information.

The application is responsible for building and parsing the SDP string. It can use the Natural Access MCC SDP library to perform these tasks. See the *SIP for Natural Call Control Developer’s Reference Manual*, for more details.

Note: The MCC SDP library is a component of the NMS SIP for NCC software.

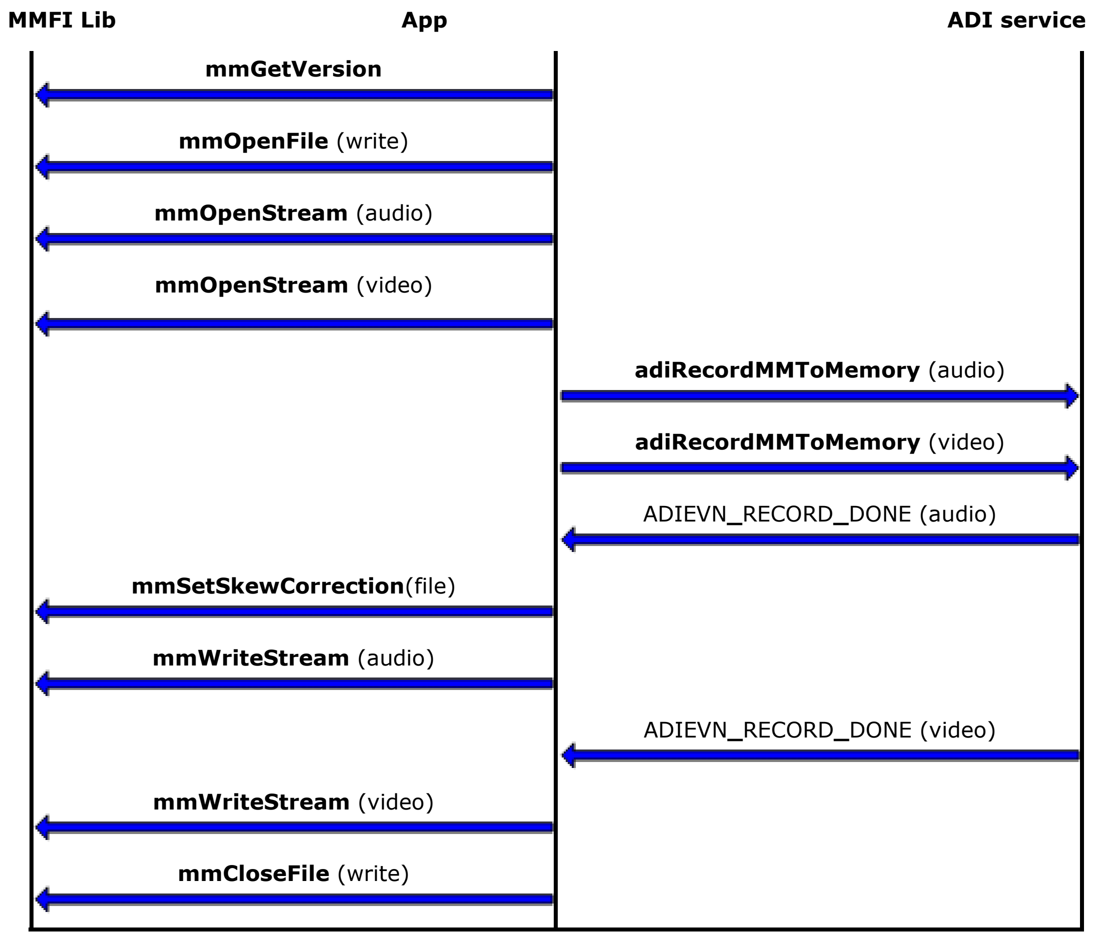
Recording to a 3GP file

To record to a 3GP file, use the Natural Access ADI service in combination with MMFI functions. Complete the following steps to record to a 3GP file:

Step	Action	Use
1	Check the library version to ensure that it supports the 3GP format.	**mmGetVersion**
2	Create the 3GP file. When you create this file, specify the format descriptor, file information, maximum file size, maximum duration, and interleaving depth.	**mmOpenFile** in write mode
3	For each type of media stream, create the media stream structure, and specify the following information: • A format of NMS packetized • AMR settings • Whether NMS native headers should be checked • Codec level • Whether a hint track should be created • MPEG4 or H.264 DCI (optional) • Whether a sync point table should be created during record	**mmOpenStream**
4	Record the NMS packetized or raw audio data into memory-resident buffers. You can record multiple streams at the same time.	**adiRecordMMToMemory** **adiRecordMMAsync**
5	Set the skew correction time if necessary. The skew correction time is handled automatically by the MMFI during the subsequent **mmWriteStream** operations.	**mmSetSkewCorrection**
6	Store the buffers as the application receives them in ADIEVN_RECORDING_DONE events. If the application is using partial buffer mode (**adiRecordMMAsync**), repeat the **mmWriteStream** call for each received event.	**mmWriteStream**
7	Close the 3GP file.	**mmCloseFile**

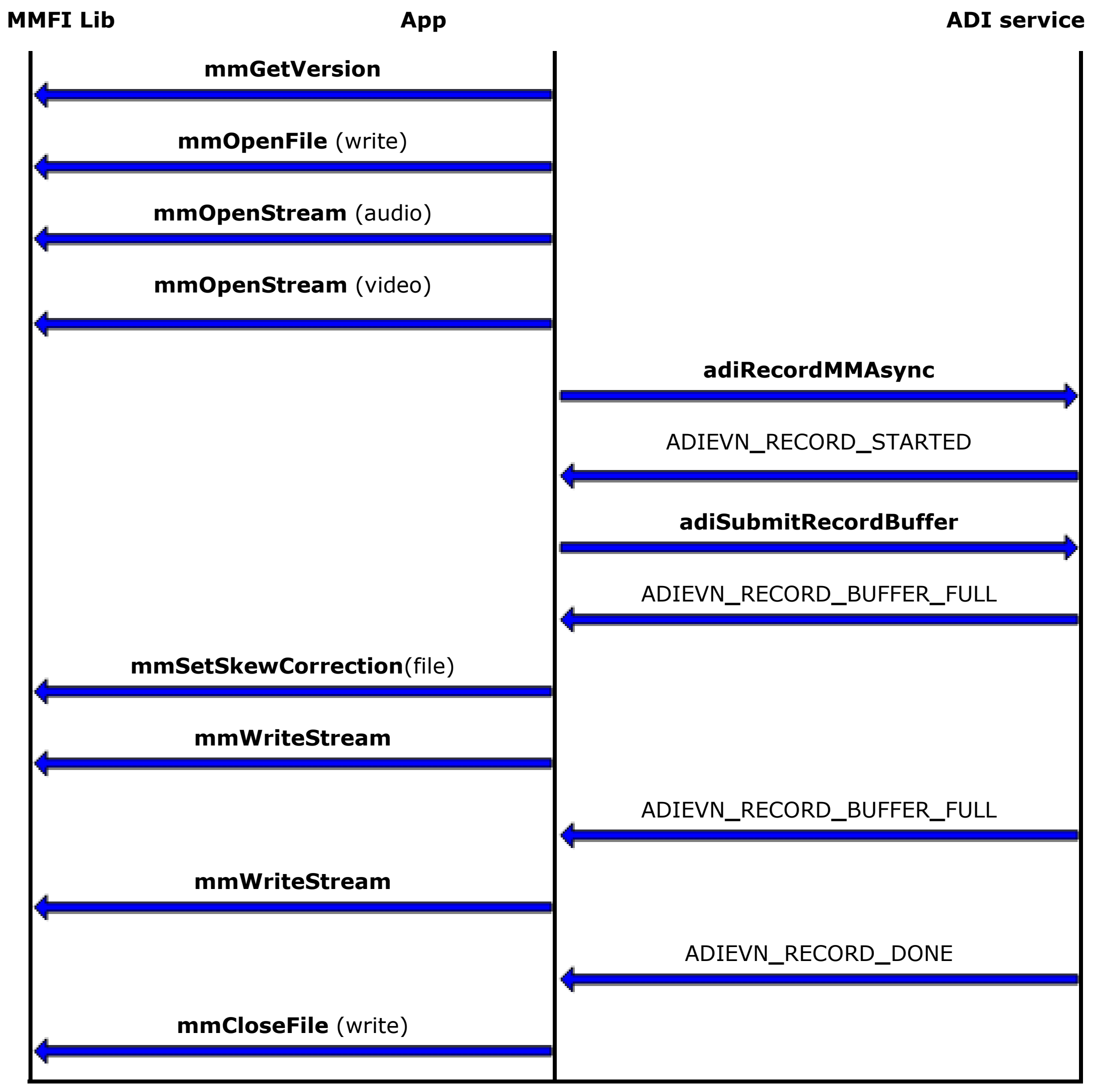
Single buffer record call flow

The call flow for a single buffer record is shown below:


MMFI Lib
App
ADI service
mmGetVersion
mmOpenFile (write)
mmOpenStream (audio)
mmOpenStream (video)
adiRecordMMToMemory (audio)
adiRecordMMToMemory (video)
ADIEVN_RECORD_DONE (audio)
mmSetSkewCorrection(file)
mmWriteStream (audio)
ADIEVN_RECORD_DONE (video)
mmWriteStream (video)
mmCloseFile (write)

Partial buffer record call flow

The call flow for a partial buffer record is shown below:



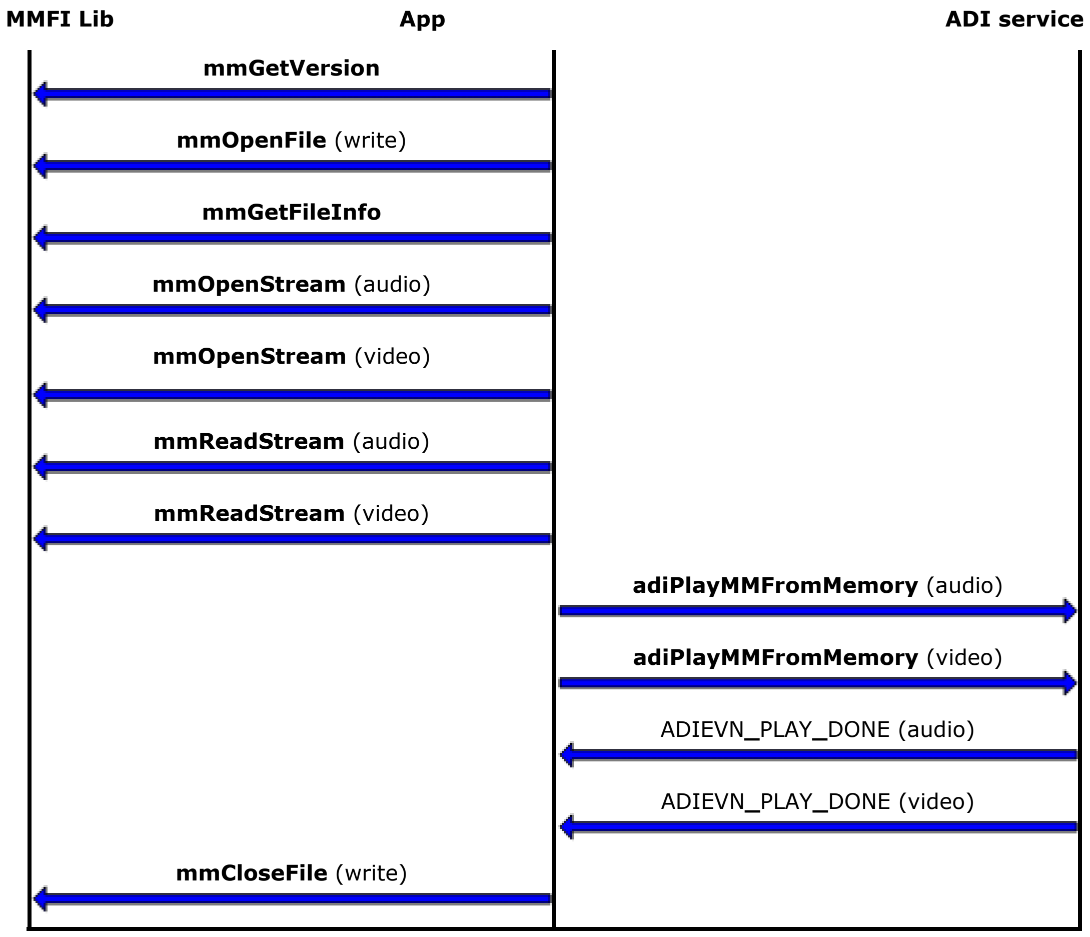
Playing a 3GP file

To play a 3GP file, use the Natural Access ADI service in combination with MMFI functions. Complete the following steps to play a 3GP file:

Step	Action	Use
1	Check the library version to ensure that it supports the 3GP format.	**mmGetVersion**
2	Open the 3GP file.	**mmOpenFile** in read mode
3	Obtain information about the 3GP file, including general file information, stream characteristics, and codec configuration.	**mmGetFileInfo**
4	For each type of media stream, create the media stream structure, and specify the following information: • The stream ID (recommended) • A format of NMS packetized (or raw format for an audio stream) • Codec information (if you do not specify the stream ID) • RTP header settings • Whether hint tracks will be used for play processing.	**mmOpenStream**
5	Read the next memory buffer from the 3GP file.	**mmReadStream**
6	Issue an ADI request to play the data. You can issue one play request for each stream type. If the application is using partial buffer mode (**adiPlayMMAsync**), repeat the **mmReadStream** call.	**adiPlayMMFromMemory** **adiPlayMMAsync**
7	When the transmission of all streams is complete, close the 3GP file.	**mmCloseFile**

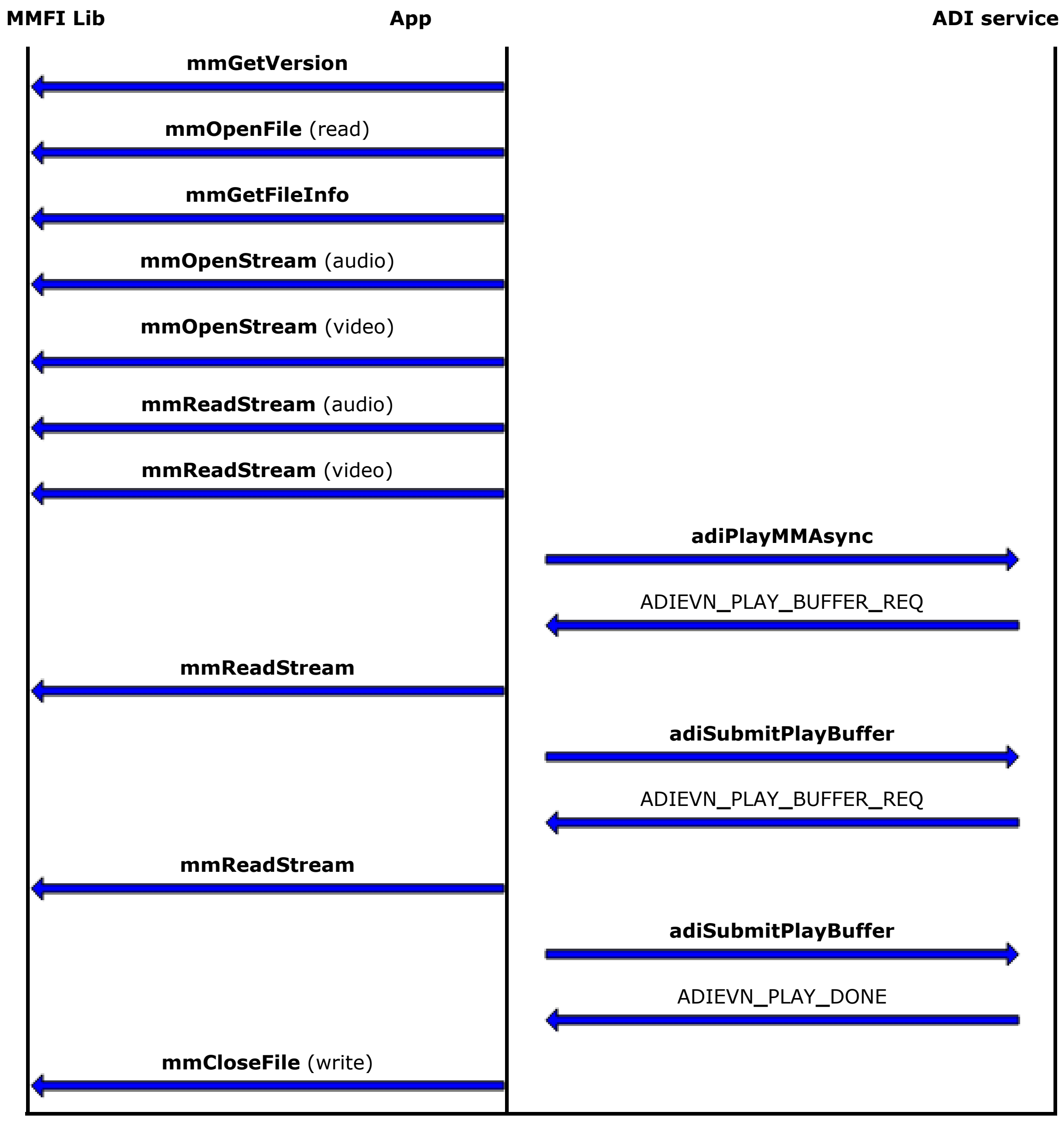
Single buffer play call flow

The call flow for a single buffer play is shown below:



Partial buffer play call flow

The call flow for a partial buffer play is shown below:



Random access call flow

To randomly access and play a 3GP file, use the Natural Access ADI service in combination with MMFI functions as explained in the examples in this section.

Restrictions on ADI Processing and random access

There are restrictions on ADI processing when performing seek operations for random access. The restrictions are:

- During a playback, seek operations (**mmSeekToTime**, **mmSeekToNextSyncPoint**, **mmSeekToPrevSyncPoint**) may not be performed while ADI play (**adiPlayMMAsync**) is in progress.
- A seek operation may only be performed when playing back using partial buffer mode **(adiPlayMMAsync**).
- Only **mmSeekToTime** and **mmSeekToNextSyncPoint** seek operations are allowed before the first ADI play function.
- When a seek operation is called after the first ADI play, the following steps must be performed:

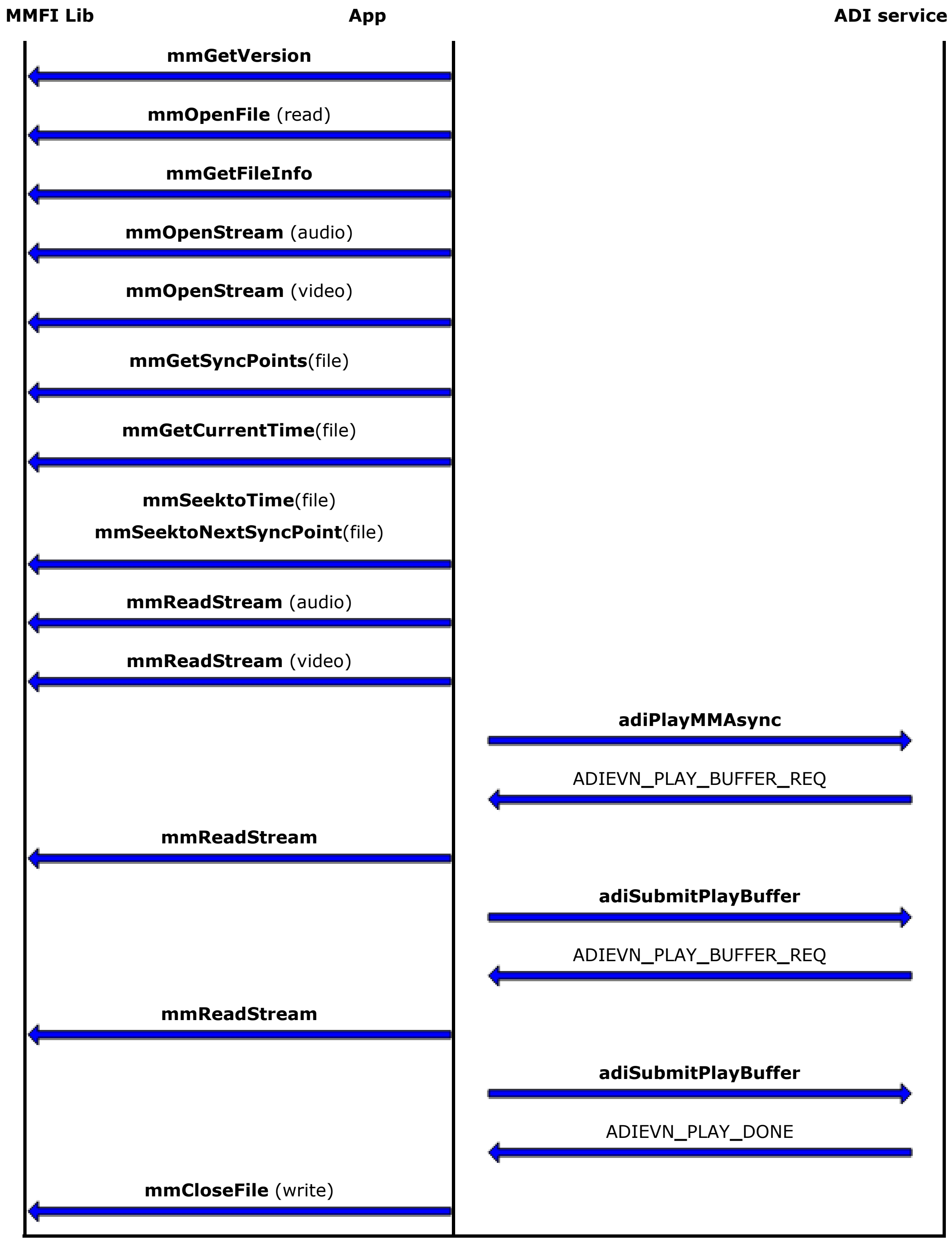
Step	Action
1	Stop ADI processing using **adiStopPlaying**.
2	Perform the seek operation using **mmSeekToTime**, **mmSeekToNextSyncPoint**, or **mmSeekToPrevPoint.**
3	Start the play process as before using **mmReadStream** and **adiStartPlaying**.

Example 1: Seeking before the first ADI play

The following table shows a call flow for using random access functions after the first ADI play function call.

Step	Action	Use
1	Check the library version to ensure that it supports the 3GP format.	**mmGetVersion**
2	Open the 3GP file.	**mmOpenFile** in read mode
3	The returned structure *FILE_INFO_DESC* on page 135 contains a field named NbSyncPoints. This provides the size of the sync table for future use in the application.	**mmGetFileInfo**
4	Open the media stream.	**mmOpenStream**
5	Allocate memory for the time-based sync table according to the field NbSyncPoints. The memory allocation is for an array of unsigned integers with size equal to NbSyncPoints.	
6	Obtain the time-based sync table. The timing information is in milliseconds.	**mmGetSyncPoints**
7	Using the obtained sync point information, set the track position pointer to a time within the 3GP file, or move the track position pointer to the next sync point.	**mmSeekToTime** **mmSeekToNextSyncPoint**
8	Read the next memory buffer from the 3GP file.	**mmReadStream**
9	Play the next buffer.	**adiPlayMMAsync**
10	When the transmission of all streams is complete, close the 3GP file.	**mmCloseFile**

A call flow using random access functions in this manner is shown below:



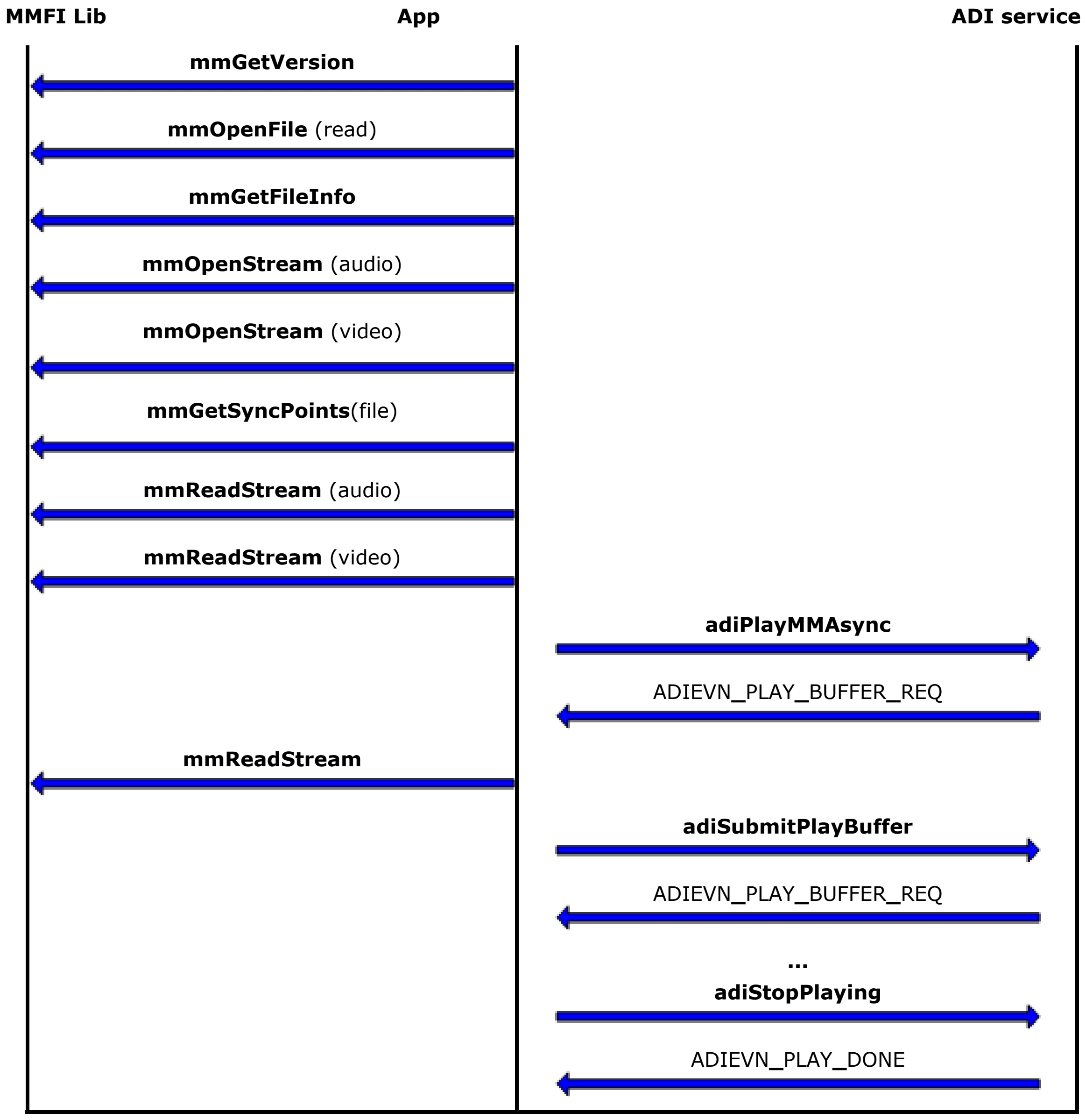
Example 2: Seeking after the first ADI play

The following table shows a call flow for using random access functions after the first ADI play function call.

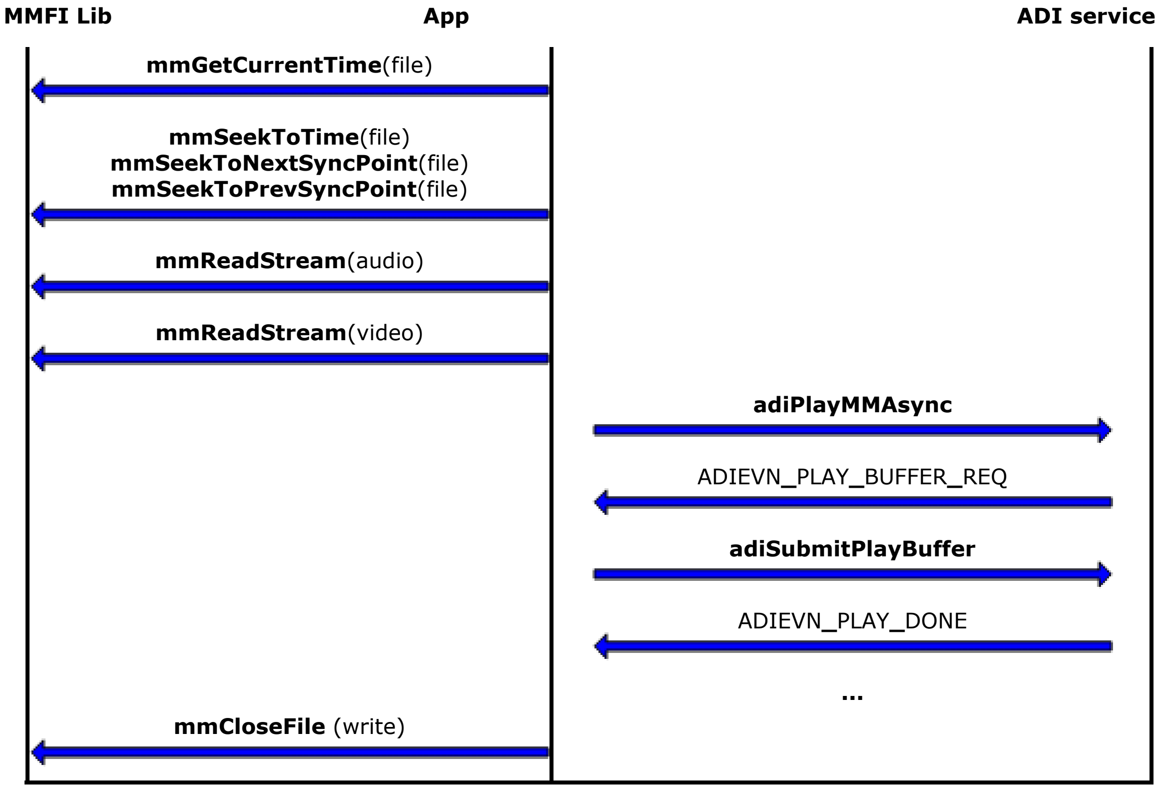
Step	Action	Use
1	Check the library version to ensure that it supports the 3GP format.	**mmGetVersion**
2	Open the 3GP file.	**mmOpenFile** in read mode
3	The returned structure *FILE_INFO_DESC* on page 135 contains a field named NbSyncPoints. This provides the size of the sync table for future use in the application.	**mmGetFileInfo**
4	Open the media stream.	**mmOpenStream**
5	Allocate memory for the time-based sync table according to the field NbSyncPoints. The memory allocation is for an array of unsigned integers with size equal to NbSyncPoints.	
6	Obtain the time-based sync table. The timing information is in units of ms.	**mmGetSyncPoints**
7	Read the next memory buffer from the 3GP file.	**mmReadStream**
8	Play the next buffer. ADI play and **mmReadStream** operations continue until the seek operation is performed.	**adiPlayMMAsync** **adiSubmitPlayBuffer**
9	Stop ADI processing before the seek operation.	**adiStopPlaying**
10	Obtain the track position pointer's current value if needed by the application.	**mmGetCurrentTime**
11	Using the obtained sync point information, set the track position pointer to a time within the 3GP file, or move the track position pointer to the next or previous sync point.	**mmSeekToTime** **mmSeekToNextSyncPoint** **mmSeekToPrevSyncPoint**
12	Read the next memory buffer from the 3GP file (as in Step 7).	**mmReadStream**
13	Play the buffer.	**adiPlayMMAsync** **adiSubmitPlayBuffer**
14	When the transmission of all streams is complete, close the 3GP file.	**mmCloseFile**

The following illustration shows a call flow performing random access on a 3GP file after the first ADI play function call:

Part 1



Part 2


MMFI Lib
App
ADI service
mmGetCurrentTime(file)
mmSeekToTime(file)
mmSeekToNextSyncPoint(file)
mmSeekToPrevSyncPoint(file)
mmReadStream(audio)
mmReadStream(video)
adiPlayMMAsync
ADIEVN_PLAY_BUFFER_REQ
adiSubmitPlayBuffer
ADIEVN_PLAY_DONE
...
mmCloseFile (write)

8 Multimedia File Interface library function summary

Multimedia file functions

Use the following functions to work with multimedia files:

Function	Description
mmCloseFile	Closes an opened multimedia file and frees intermediate internal resources. If the file was opened in write mode, then **mmCloseFile** also writes data.
mmGetFileInfo	Obtains detailed information about an opened multimedia file.
mmOpenFile	Opens a multimedia file for read or write operations.

3GP file functions

Use the following functions to randomly access the contents of a 3GP file during a play operation:

Function	Description
mmGetCurrentTime	Obtains the current time elapsed during a play operation.
mmGetSyncPoints	Obtains sync point table information in the file.
mmSeekToNextSyncPoint	Seeks to the next sync point in the file.
mmSeekToPrevSyncPoint	Seeks to the previous sync point in the file.
mmSeekToTime	Seeks to a time relative to the start of the file.

Skew correction functions

Use the following function to set the skew correction in a multimedia file:

Function	Description
mmSetSkewCorrection	Sets the skew correction value in the multimedia file.

SDP functions

Use the following functions to store/retrieve SDP data to/from a multimedia file:

Function	Description
mmSetSDPInfo	Sets the SDP information in the multimedia file.
mmGetSDPInfo	Gets the SDP information from the multimedia file.

Note: These functions may be used as multimedia file functions or multimedia stream functions.

Multimedia stream functions

Use the following functions to work with multimedia streams:

Function	Description
mmOpenStream	Opens a stream in a multimedia file for read or write operations.
mmReadStream	Reads media data out of a stream at the current position.
mmWriteStream	Writes media data either into an intermediate holding area or directly into the specified stream at the current position.

SDP functions

Use the following functions to store/retrieve SDP data to/from a multimedia file:

Function	Description
mmSetSDPInfo	Sets the SDP information in the multimedia file.
mmGetSDPInfo	Gets the SDP information from the multimedia file.

Note: These functions may be used as multimedia file functions or multimedia stream functions.

Version information function

Use **mmGetVersion** to obtain the version and supported formats of the MMFI library.

9 Multimedia File Interface library functions

Using the MMFI library function reference

The function reference describes each MMFI function and displays its prototype, return values, and associated events (if any). It provides additional details, when necessary, and also provides a code sample for the function.

Prototype	The prototype is followed by a listing of the function's arguments. Each of the MMFI library functions has the DWORD (16-bit unsigned) data type. If a function uses a structure, the structure is listed as an argument.
Return values	A return value of SUCCESS (0) indicates the function was initiated; subsequent events indicate the status of the operation. Other possible return values are listed in the description of each individual call.
Events	MMFI library events are returned in the application's event buffer. Additional information such as reason codes and return values may be provided in the value field of the event.
Example	Example functions are taken from sample application programs shipped with the product. The notation /* ... */ indicates additional code that is not shown.

mmCloseFile

Closes an opened multimedia file and frees intermediate internal resources. If the file was opened in write mode, then **mmCloseFile** also writes data, as described in the Details section.

Prototype

DWORD **mmCloseFile** (MMFILE ***pMMFile**)

Argument	**Description**
pMMFile	A pointer to the object that controls the multimedia file.

Return values

Return value	**Description**
SUCCESS	
MMERR_FILE_WRITE_FAILURE	File write failure; discard the file.
MMERR_INTERNAL_RESOURCE	Error on internal resource.
MMERR_INVALID_FILE	***pMMFile*** is invalid.

Details

If the file was opened in write mode, **mmCloseFile** writes the data in the control header. **mmCloseFile** also writes data to the media stream, if the file was opened with one of these format settings**:**

- FORMAT_FLAG_WR_NORMAL
- FORMAT_FLAG_TEMP_MEM
- FORMAT_FLAG_TEMP_FILE

If the file was opened with the FORMAT_FLAG_WR_DIRECT setting, then the data is written to the media stream when **mmWriteStream** is invoked. For more information about opening a file, see **mmOpenStream**.

Example

```
DWORD ret;
ret = mmCloseFile(&mmFile);
if( ret != SUCCESS )
{
    printf("ERROR: cannot close 3gp file\n");
    return -1;
}
```

mmGetCurrentTime

Obtains the current time elapsed during a play operation.

Prototype

DWORD **mmGetCurrentTime** (MMFILE ***pMMFile**, unsigned int ***currTime**)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
currTime	Returns the current time, based on the video track time, in milliseconds.

Return values

Return value	Description
SUCCESS	
MMERR_NO_SYNC_TABLE	File does not have a sync table.
MMERR_INVALID_FILE	pMMFile is NULL.

Details

mmGetCurrentTime obtains the current time elapsed during a play operation. The current time is useful information for seek operations.

For example, to seek ahead 30 seconds from the current time, the application:

1. Gets the current time using **mmGetCurrentTime**.
2. Adds the seek ahead time (30 seconds) to the current time.
3. Uses **mmSeekToTime** to seek to the computed time.

Example

```
DWORD ret;
Unsigned int currTime;
ret = mmGetCurrentTime(&mmFile, &currTime);
If( ret != SUCCESS )
{
printf ("\nERROR in mmGetSyncPoints (%d)\n", ret);
return -1;
}
```

mmGetFileInfo

Obtains detailed information about an opened multimedia file.

Prototype

DWORD **mmGetFileInfo** (MMFILE ***pMMFile,** FILE_INFO_DESC ***pFileInfoDesc**)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.

pFileInfoDesc	A pointer to the returned FILE_INFO_DESC data structure that is the descriptor for the multimedia file. The structure definition is:
	`typedef struct`
	`    {`
	`    WORD fileInfoSize;`
	`    BYTE *pFileInfo;`
	`    WORD fileType;`
	`#define FILE_TYPE_UNKWN 0`
	`#define FILE_TYPE_3GP 1`
	`#define FILE_TYPE_MP4 2`
	`    char format [8];`
	`    DWORD version;`
	`    DWORD flags;`
	`#define DESC_FLAG_INSUFF_SPACE 1`
	`    WORD  NbSyncPoints;`
	`    WORD  reserved [5];`
	`} FILE_INFO_DESC;`
	For information about the fields in this structure, see FILE_INFO_DESC.

Return codes

Return value	Description
SUCCESS	
MERR_BUFFER_TOO_SMALL	File information descriptor is too small to receive the minimum data.
MMERR_FILE_FORMAT	Error in file format.
MMERR_FILE_READ_FAILURE	File read failure.
MMERR_INCOMP_ACCESS_MODE	File was not opened for read operation.
MMERR_INVALID_FILE	***pMMFile*** is not valid.
MMERR_INVALID_POINTER	***pFileInfoDesc*** is NULL.

Details

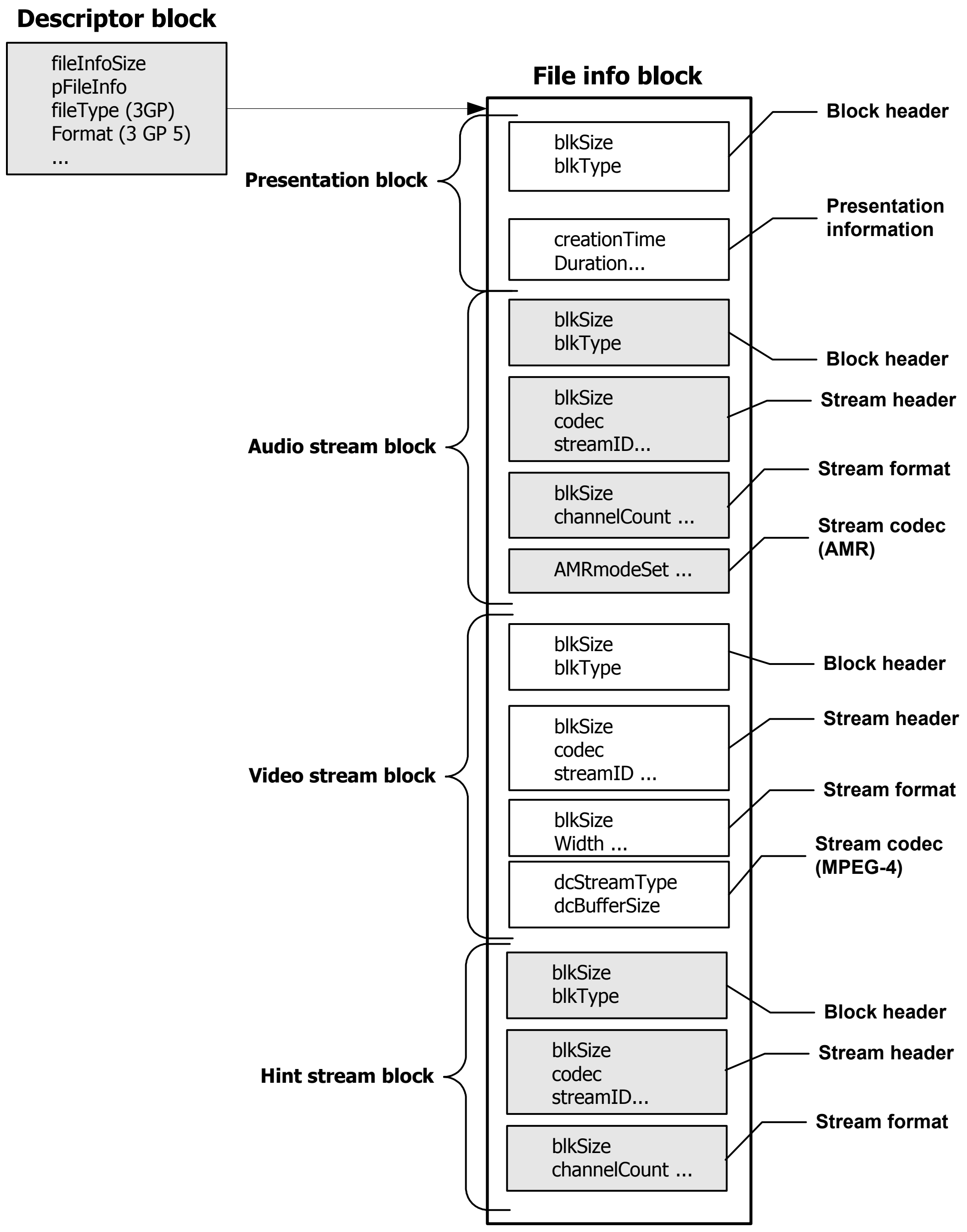
mmGetFileInfo supports 3GP files in Basic Profile and Streaming Profile. To use **mmGetFileInfo**, first invoke **mmOpenFile** in read mode.

mmGetFileInfo fills in the descriptor block (*FILE_INFO_DESC* on page 135 structure) and initializes the buffer that contains detailed information about the file (pFileInfo). This block of data is called the file info block. The format of the file info block depends on the type of file indicated in the descriptor block. It consists of one, or more of the following types of information blocks:

- Presentation block (always present when **mmGetFileInfo** returns SUCCESS).
- Audio stream block (present if an audio track is in the file).

- Video stream block (present if a video track is in the file).
- Hint stream block (present if a hint track is in the file. A block is present for each hint track created).

The following illustration shows the layout of the file info block:



Note: If the format of the file is not 3GP-Basic Profile compliant, only the descriptor is initialized.

Parsing a 3GP file

Because information blocks can be different sizes and types, each information block has a header in the beginning of the block that contains size and type information. Parse the header information by using the FILE_INFO_BLOCK_HEADER structure. Parse the remaining part of the block according to the block's type, which is indicated by the value of the blkType field.

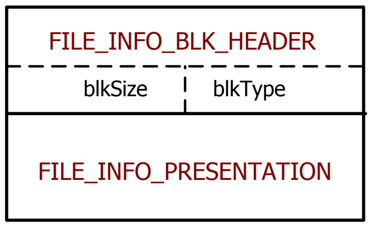
- If the block is a presentation block, the value of the blkType field is BLK_TYPE_PRESENTATION.
- If the block is a stream block, the value of the blkType field is BLK_TYPE_STREAM.

Parsing a presentation block

To parse a presentation block:

Step	**Action**
1	Determine the block size by viewing the blkSize field in the FILE_INFO_BLOCK_HEADER structure.
2	Parse the presentation block information according to the FILE_INFO_PRESENTATION structure.

The following illustration shows the components of a presentation block:



Parsing a stream block

To parse a stream block:

Step	**Action**
1	Determine the block size by viewing the blkSize field in the FILE_INFO_BLOCK_HEADER structure.
2	Parse the FILE_INFO_STREAM_HEADER structure to view stream information that is common to audio and video streams. The value of the streamType field in this structure indicates whether the stream block contains audio stream information (STREAM_TYPE_AUDIO) or video stream information (STREAM_TYPE_VIDEO). The value of the codec field indicates the kind of codec-specific information that the stream block contains.
3	Parse an audio stream according to the FILE_INFO_FORMAT_AUDIO structure, and a video stream according to the FILE_INFO_FORMAT_VIDEO structure.
4	Parse the codec-specific information according to FILE_INFO_CODEC_AMR, FILE_INFO_CODEC_H263, FILE_INFO_CODEC_H264, or FILE_INFO_CODEC_MPEG4.

Example

```
FILE_INFO_DESC fileInfoDesc
FILE_INFO_PRESENTATION *pPresentation;
ret = mmGetFileInfo (pMMFile, &fileInfoDesc);
pPresentation         = (FILE_INFO_PRESENTATION *)fileInfoDesc.pFileInfo;
blockSize             = pPresentation->blockHeader.blkSize;
streamCount           = pPresentation -> streamCount;
for (j = 0; j < streamCount; j++)
    {
        blockSize     = ((FILE_INFO_BLOCK_HEADER *)pFileInfo) -> blkSize;
        ....
        pFileInfo     = blockSize;
    }
```

mmGetSDPInfo

Obtains the SDP data from a 3GP file.

Prototype

DWORD **mmGetSDPInfo** (MMFILE ****pMMFile***, BYTE ****pBuffer***, unsigned ****pBufferSize***)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
pBuffer	A pointer to the SDP data buffer.
pBufferSize	A pointer to the SDP data buffer's size field.

Return values

Return value	Description
SUCCESS	
MMERR_INVALID_FILE	***pMMFile*** is NULL.

Details

After calling **mmGetSDPInfo** the SDP buffer contains session-level SDP parameters followed by media-level parameters.

mmGetSDPInfo should be called after the final **mmOpenStream**.

The application allocates memory for the SDP data buffer using the value of the sdpBufferSize field in the FILE_INFO_DESC structure returned from **mmGetFileInfo**.

Examples

```
DWORD ret;
ret = mmGetSDPInfo(&mmFile, &mmStream, pBuffer, pBufferSize);
If( ret != SUCCESS )
{
printf ("\nERROR in mmGetSDPInfo(%d)\n", ret);
return -1;
}
```

mmGetSyncPoints

Obtains sync point information from the sync point table in a multimedia file.

Prototype

DWORD **mmGetSyncPoints** (MMFILE ***pMMFile**, int ***pSyncTable**)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
pSyncTable	A pointer to the start of the application's sync table.

Return values

Return value	Description
SUCCESS	
MMERR_NO_SYNC_TABLE	File does not have a sync table.
MMERR_INVALID_FILE	pMMFile is NULL.

Details

mmGetSyncPoints gets the list of times associated with the 3GP file's sync points and updates the pre-allocated sync table with these times. The application allocates memory for the time-based sync table using the value of the NbSyncPoints field in the FILE_INFO_DESC structure returned from **mmGetFileInfo**.

Examples

```
DWORD ret;
ret = mmGetSyncPoints(&mmFile, pSyncTable);
If( ret != SUCCESS )
{
printf ("\nERROR in mmGetSyncPoints (%d)\n", ret);
return -1;
}
```

mmGetVersion

Obtains the version and supported formats of the Multimedia File Interface library.

Prototype

DWORD **mmGetVersion** (MMFI_VERS_INFO ***pInfo***, WORD ***infoSize***)

Argument	Description
pInfo	A pointer to the MMFI_VERS_INFO structure that contains information about the version and supported formats of the Multimedia File Interface library.
infoSize	The size of the MMFI_VERS_INFO structure.

Return values

Return value	Description
SUCCESS	
MMERR_BUFFER_TOO_SMALL	***infoSize*** is too small to receive the information structure.
MMERR_INVALID_POINTER	***pInfo*** is NULL.

Example

```
MMFI_VERS_INFO mmFileVersInfo;
WORD fileVersInfoSize = sizeof(mmFileVersInfo);
DWORD ret;

if ((ret = mmGetVersion(&mmFileVersInfo, fileVersInfoSize)) != SUCCESS)
    {
        printf ("\nERROR in mmGetVersion (%d)\n", ret);
        exit( 1 );
    }
```

mmOpenFile

Opens a multimedia file for read or write operations.

Prototype

DWORD **mmOpenFile** (char ***pFileName,*** unsigned ***openMode***, unsigned ***fileType***, FILE_FORMAT_DESC ****pFileFormatDesc***, unsigned ***verboseLevel***, MMFILE ****pMMFile***)

Argument	Description
pFileName	A pointer to a text string that contains the name of the 3GP file.
openMode	Specifies whether the file will be opened for reading or writing. Valid values are: • OPEN_MODE_READ - Opens a file for reading. • OPEN_MODE_WRITE - Open a file for writing. If the file exists, its contents are destroyed.
fileType	Determines the type of file; and therefore, the file handler. The only valid value is FILE_TYPE_3GP. (File is a 3GP file.)
pFileFormatDesc	(Write mode only) A pointer to the FILE_FORMAT_DESC data structure that determines the format of the created file. A value of NULL specifies that the file will be created with the default format. The structure definition is: `typedef struct { char format [8]; DWORD version; DWORD flags; DWORD maxFileSize; DWORD maxFileTime; DWORD interleave; } FILE_FORMAT_DESC;` For information about the fields in this structure, see FILE_FORMAT_DESC.
verboseLevel	Controls the trace information printed on stdoutput. Valid values are: • VERBOSE_NOTRACES_LEVEL - No printouts. • VERBOSE_INFO_LEVEL - Full informative printouts. • VERBOSE_DEBUG_LEVEL - Debug printouts. • VERBOSE_WARNING_LEVEL - Warning printouts. • VERBOSE_ERROR_LEVEL - Error-level printouts.
pMMFile	A pointer to the object that controls the multimedia file.

Return values

Return value	Description
SUCCESS	
MMERR_FILE_ACCESS_DENIED	Access denied in specified open mode.
MMERR_FILE_FORMAT	Error in file format.
MMERR_FILE_OPEN_FAILED	File was not found.
MMERR_INTERNAL_RESOURCE	Error on internal resource.
MMERR_INVALID_FILE_TYPE	Specified file type is not supported.
MMERR_INVALID_FORMAT_DESC	Invalid field in format descriptor.
MMERR_INVALID_OPEN_MODE	Invalid open mode.
MMERR_INVALID_POINTER	***pMMFile or pFileName*** is NULL.

Details

Use **mmOpenFile** to open a multimedia file in read mode or write mode.

Opening a file in read mode

Opening a file in read mode gives you read access to the existing file. After you open the file, you can invoke **mmGetFileInfo** to obtain detailed information about the file.

Opening a file in write mode

Opening a file in write mode allows you to create a new file or overlay an existing file. If you open an existing file in write mode, the file contents are automatically destroyed.

To update the opened file, make changes to the returned MMFILE structure by using **mmWriteStream**. Depending on the format settings you specify when you open the file, the data either gets written to the media stream every time you invoke **mmWriteStream**, or it gets written later on, when you close the file.

Examples

Opening a 3GP file for read mode

```
DWORD ret;
insigned verboseLevel = 0;
MMFILE mmFile;
ret = mmOpenFile( “record.3gp”, OPEN_MODE_READ, FILE_TYPE_3GP,
                  NULL, verboseLevel, &mmFile);
If( ret != SUCCESS )
{
    printf ("\nERROR in mmOpenFile (%d)\n", ret);
    return -1;
}
```

Opening a 3GP file for write mode

```
DWORD ret;
insigned verboseLevel = 0;
MMFILE mmFile;
FILE_FORMAT_DESC fileFormatDesc;
ret = mmOpenFile( “record.3gp”, OPEN_MODE_READ, FILE_TYPE_3GP,
                  &fileFormatDesc, verboseLevel, &mmFile);
If( ret != SUCCESS )
{
    printf ("\nERROR in mmOpenFile (%d)\n", ret);
    return -1;
}
```

mmOpenStream

Opens a stream in a multimedia file for read or write operations.

Prototype

DWORD **mmOpenStream** (MMFILE ***pMMFile**, DWORD ***streamID***, WORD ***streamType***, WORD ***codec***, DATA_FORMAT_DESC ***pDataFormatDesc***, MMSTREAM ***pMMStream***, DATA_FORMAT_INFO ***pDataFormatInfo***)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
streamID	The ID of the stream to open. For a: • Read operation, a value of 0 (zero) specifies that the first stream found of a given type and codec should be opened. • Write operation, this parameter is ignored, and the stream ID is set automatically. NMS Communications recommends setting this value to NULL for a write operation.
streamType	The type of stream to open. Valid values for ***streamType*** depend on the mode:

Mode	Valid values
Read	Valid values are: • STREAM_TYPE_UNKNWN - An unsupported or unrecognized stream. • STREAM_TYPE_AUDIO - An audio stream. • STREAM_TYPE_VIDEO - A video stream. If you specify STREAM_TYPE_UNKNWN, the codec is ignored and the stream is selected with the stream ID.
Write	Valid values for ***streamType*** include all of the values listed for Read mode, except for STREAM_TYPE_UNKNWN. You must explicitly specify the stream type, if you invoked the function in write mode.

Argument	Description
codec	The codec of the stream to open. Valid values for ***codec*** depend on the mode.

Mode	Valid values
Read	Valid values are: • S_CODEC_AMR - Audio stream AMR codec. • S_CODEC_H263 - Video stream, H.263 codec. • S_CODEC_H264 - Video stream, H.264 codec. • S_CODEC_MPEG4 - Video stream, MPEG-4 codec. • S_CODEC_UNKNWN - An unsupported or unrecognized codec. If you specify STREAM_TYPE_UNKNWN, the codec is ignored, and the stream is selected by the stream type and ID.
Write	Valid values include all of the values listed for Read mode, except for S_CODEC_UNKNWN. You must explicitly specify the codec for a write operation.

Argument	Description
pDataFormatDesc	A pointer to the DATA_FORMAT_DESC structure that determines the format of the data, including whether it is raw or NMS packetized. A NULL value indicates that the default format (NMS_PACKETIZED) is used. The structure definition is: (see below)
pMMStream	A pointer to the returned MMSTREAM data structure that is used to access the multimedia stream. The structure definition is: (see below)
pDataFormatInfo	An optional pointer to the DATA_FORMAT_INFO structure that provides size characteristics for the opened stream. The structure definition is: (see below)

pDataFormatDesc

A pointer to the DATA_FORMAT_DESC structure that determines the format of the data, including whether it is raw or NMS packetized. A NULL value indicates that the default format (NMS_PACKETIZED) is used.

The structure definition is:

```
typedef struct
{
    WORD    format;
#define FORMAT_NMS_PACKETIZED 0
#define FORMAT_RAW 1 /*
    DWORD   flags; //!< Format flags (bitmap).
#define FORMAT_FLAG_AMR_IF2          0x001
#define FORMAT_FLAG_AMR_DTX          0x002
#define FORMAT_FLAG_H263_2190        0x004
#define FORMAT_FLAG_RTP_HINTS        0x008
#define FORMAT_FLAG_RTP_SETTINGS     0x0010
#define FORMAT_FLAG_HEADER_CHECK     0x0020
#define FORMAT_FLAG_MEM_CHUNKS       0x0080
#define FORMAT_FLAG_DCI_INFO       0x0100
#define FORMAT_WRITE_SYNC_POINTS     0x2000
    WORD      rtpTimestampOffset;
    WORD      rtpSequenceOffset;
    WORD      rtpPayloadType;
#define DEFAULT_PAYLOAD_TYPE 0
    WORD      rtpPayloadSize;
#define DEFAULT_PAYLOAD_SIZE 0
    BYTE      videoProfile;
    BYTE      videoLevel;
    BYTE      *pdciInfo;
    int       dciInfoSize;
WORD       videoAggregationThreshold;
} DATA_FORMAT_DESC;
```

For information about the fields in this structure, see *DATA_FORMAT_DESC* on page 130.

pMMStream

A pointer to the returned MMSTREAM data structure that is used to access the multimedia stream. The structure definition is:

```
typedef struct mm_stream
{
    unsigned              state;
    unsigned              streamID;
    void                 *msp;
    struct mm_stream     *pNext;
    struct mm_file       *pMMFile;
} MMSTREAM;
```

pDataFormatInfo

An optional pointer to the DATA_FORMAT_INFO structure that provides size characteristics for the opened stream. The structure definition is:

```
typedef struct
{
    DWORD estTotalStreamSize;
    DWORD minReadBufferSize;
} DATA_FORMAT_INFO;
```

For information about the fields in this structure, see *DATA_FORMAT_INFO* on page 133.

Return values

Return value	Description
SUCCESS	
MMERR_FILE_FORMAT	(Read mode only) Error in file format.
MMERR_FILE_READ_FAILURE	(Read mode only) File read failure.
MMERR_FILE_WRITE_FAILURE	(Write mode only) File write failure.
MMERR_INTERNAL_RESOURCE	Error on internal resource.
MMERR_INVALID_FILE	pMMFile is invalid.
MMERR_INVALID_FORMAT_DESC	Invalid field in format descriptor.
MMERR_INVALID_POINTER	pMMStream is NULL.
MMERR_INVALID_STREAM	Stream not found. The stream may already be opened, or may be incorrectly specified.
MMERR_MAX_STREAM	Maximum number of opened streams reached.
MMERR_NO_LICENSE_AVAILABLE	No license is available.

Events

None.

Details

The mode that **mmOpenStream** uses to open a stream depends on how the file containing the stream was opened.

- If you used **mmOpenFile** in read mode to open the file, then the stream is also opened in read mode.
- If you used **mmOpenFile** in write mode to open the file, then the stream is also opened in write mode.

For a read operation, the stream can be selected by any combination of type, codec, or ID. Typically, an application selects a stream with the ID only or with the type and codec.

Example

```
DWORD ret;
MMSTREAM mmStream;
DATA_FORMAT_INFO dataFormatInfo;
DATA_FORMAT_DESC dataFormatDesc;

streamID = 1;
streamType = STREAM_TYPE_VIDEO;
codec = S_CODEC_MPEG4;
dataFormatDesc. format = FORMAT_NMS_PACKETIZED;
dataFormatDesc. flags   = FORMAT_FLAG_AMR_IF2;
dataFormatDesc. videoLevel = 0;
dataFormatDesc. videoProfile = 0;

ret = mmOpenStream(&mmFile, streamID,  streamType, codec, &dataFormatDesc,
                   &mmStream, &dataFormatInfo);

if( ret != SUCCESS )
{
    printf ("\nERROR in mmOpenStream for stream ID (%lu), ret=(%d)\n",
    streamID, ret );
    return -1;
}
```

mmReadStream

Reads media data out of a stream at the current position. Only entire media samples are read. Media samples are also called frames for audio and video data.

Prototype

DWORD **mmReadStream** (MMSTREAM ***pMMStream***, unsigned ***sampleCount***, BYTE ***pBuffer***, unsigned ***bufferSize***, unsigned ***pByteCount***)

Argument	Description
pMMStream	Returns a pointer to the object that controls the stream.
sampleCount	The number of samples to read. Specify SAMPLE_COUNT_MAX to read as many entire samples as the buffer can contain.
pBuffer	A pointer to the recipient buffer where the media data is copied.
bufferSize	The maximum size of the recipient buffer, in bytes.
pByteCount	A pointer to the actual number of bytes copied. The Video Messaging Server Interface returns this value after the end of stream is reached (return value MMERR_END_OF_STREAM).

Return values

Return value	Description
SUCCESS	
MMERR_BUFFER_TOO_SMALL	Recipient buffer is too small to receive the entire media sample.
MMERR_END_OF_STREAM	End of stream reached. Check the ***pByteCount*** value.
MMERR_FILE_FORMAT	Error in file format.
MMERR_FILE_READ_FAILURE	File read failure.
MMERR_INCOMP_ACCESS_MODE	Stream was not opened for read operation.
MMERR_INVALID_POINTER	Either ***pBuffer*** or ***pByteCount*** is null.
MMERR_INVALID_STREAM	***pMMStream*** is invalid.

Example

```
DWORD ret;
unsigned char *pBuffer;
unsigned byteCount;
pBuffer = (unsigned char *)malloc(bufferSize);

ret = mmReadStream(&mmStream, SAMPLE_COUNT_MAX, pBuffer,
bufferSize, &byteCount);

if ((ret != SUCCESS) && (ret != MMERR_END_OF_STREAM))
    {
      printf("\nERROR: failed to read the stream from file. ret (%d)\n", ret);
    }
```

mmSeekToNextSyncPoint

Seeks all of the tracks in a multimedia file to the next sync point.

Prototype

DWORD **mmSeekToNextSyncPoint** (MMFILE ***pMMFile**)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.

Return values

Return value	Description
SUCCESS	
MMERR_NO_SYNC_TABLE	File does not have a sync table.
MMERR_INVALID_FILE	***pMMFile*** is NULL.
MMERR_NO_MORE_SYNC_POINTS	If the application attempts to seek to the next sync point and the current time is greater than or equal to the last sync point time in the sync table, this error is returned.

Details

Use **mmSeekToNextSyncPoint** to seek to the next sync point in the file. The track pointer for all tracks is repositioned to the next sync point time. **mmSeekToNextSyncPoint** can only be called during a play operation being done in partial buffer mode.

Examples

```
DWORD ret;
ret = mmSeekToNextSyncPoint(&mmFile);
If( ret != SUCCESS )
{
printf ("\nERROR in mmSeekToNextSyncPoint (%d)\n", ret);
return -1;
}
```

mmSeekToPrevSyncPoint

Seeks all of the tracks in a multimedia file to the previous sync point.

Prototype

DWORD **mmSeekToPrevSyncPoint** (MMFILE ***pMMFile)***

Argument Description

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.

Return values

Return value	Description
SUCCESS	
MMERR_NO_SYNC_TABLE	File does not have a sync table.
MMERR_INVALID_FILE	***pMMFile*** is NULL.
MMERR_NO_PREV_SYNC_POINT	If the application attempts to seek to a previous sync point and the current time is less than or equal to the first sync point time in the sync table, this error is returned.

Details

Use **mmSeekToPrevSyncPoint** to seek to the previous sync point in the file. The track pointer for all tracks is repositioned to the previous sync point time. **mmSeekToPrevSyncPoint** can only be called during a play operation being done in partial buffer mode.

Examples

```
DWORD ret;
ret = mmSeekToPrevSyncPoint(&mmFile);
If( ret != SUCCESS )
{
printf ("\nERROR in mmSeekToPrevSyncPoint (%d)\n", ret);
return -1;
}
```

mmSeekToTime

Sets all tracks in a multimedia file to a specific time during a play operation.

Prototype

DWORD **mmSetSeekTime** (MMFile ****pMMFile***, unsigned int ***seekTime***)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
seekTime	A specific track time within a file relative to the beginning of the file, specified in milliseconds.

Return values

Return value	Description
SUCCESS	
MMERR_NO_SYNC_TABLE	File does not have a sync table.
MMERR_INVALID_FILE	***pMMFile*** is NULL.
MMERR_INVALID_TIME	The ***seekTime*** is invalid. This error is returned if either of the following is true: • ***seekTime*** is greater than the last sync point time. • ***seekTime*** is less than the first sync point time.

Details

Use **mmSeekToTime** to seek to a specified time relative to the start of a file. The track pointer for all tracks is repositioned to the specified time. **mmSeekToTime** can be called only during a play operation. A seek operation can be performed when using partial buffer mode (not single buffer mode).

When the application calls **mmSeekToTime** on a time that does not match a sync point, the track pointer is repositioned to the next seek point's time.

Example

```
DWORD ret;
ret = mmSeekToTime(&mmFile, seekTime);
If( ret != SUCCESS )
{
    printf ("\nERROR in mmSeekToTime (%d)\n", ret);
    return -1;
}
```

mmSetSDPInfo

Set the SDP data from a 3GP file.

Prototype

DWORD **mmSetSDPInfo** (MMFILE ****pMMFile***, BYTE ****pBuffer***, unsigned ***bufferSize***)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
pBuffer	A pointer to the SDP data buffer.
bufferSize	SDP data buffer's size field.

Return values

Return value	Description
SUCCESS	
MMERR_INVALID_FILE_AND_STREAM	***pMMFile*** is NULL.

Details

mmSetSDPInfo stores the SDP data to a 3GP file. The SDP buffer should contain session level SDP parameters followed by media level parameters.

mmSetSDPInfo should be called after the final **mmOpenStream**.

Examples

```
DWORD ret;
ret = mmGetSDPInfo(&mmFile, pBuffer, pBufferSize);
if( ret != SUCCESS )
{
   printf ("\nERROR in mmGetSDPInfo(%d)\n", ret);
   return -1;
}
```

mmSetSkewCorrection

Sets the skew correction time for an audio stream relative to the video stream.

Prototype

DWORD **mmSetSkewCorrection** (MMFile ***pMMFile***, int ***skewCorrection***)

Argument	Description
pMMFile	A pointer to the object that controls the multimedia file.
skewCorrection	Skew correction time for an audio (AMR) stream during a 3GP file record. The effect of a skew correction is based on the fact that AMR frames are 20 ms duration:

If the skewCorrection value is...	Then...
Greater than 0 ms	The first audio frame is repeated (***n*** ms / 20 ms) times in the audio data stream. For example, if ***skewCorrection*** is 40 ms, the first audio frame is repeated twice (40 ms / 20 ms = 2).
Less than 0 ms	The first (***n*** ms / 20 ms) audio frames are removed from the audio stream. For example, if ***skewCorrection*** = -40 ms, the first two audio frames are removed from the audio stream (-40 ms / 20 ms = -2).

Valid values are -4095 to 4095 ms.

Return values

Return value	Description
SUCCESS	
MMERR_INVALID_SKEW_TIME	Skew time is not -4095 to 4095 ms.

Details

Use **mmSetSkewCorrection** to synchronize the audio and video streams. The skew correction operation is allowed only during the record of a 3GP file. Call **mmSetSkewCorrection** before the first call to **mmWriteStream** for the audio stream.

Example

```
DWORD ret;
int skewCorrection = 20; /* Set skew correction to 20 ms */
ret = mmSetSkewCorrection( &mmfile, skewCorrection);
if (ret != SUCCESS)
{
printf("\nERROR: failed to set the skew correction time ... ret (%d)\n", ret);
}
```

mmWriteStream

Writes media data either into an intermediate holding area or directly into the specified stream at the current position.

Prototype

DWORD **mmWriteStream** (MMSTREAM ***pMMStream***, BYTE ***pBuffer***, unsigned ***bufferSize***, unsigned ***writeFlags***, unsigned ***pByteCount***, unsigned ***pSampleIdx***, unsigned ***pTime***)

Argument	Description
pMMStream	Returns a pointer to the object that controls the stream.
pBuffer	A pointer to the buffer that contains the media data to write.
bufferSize	The size of the buffer, in bytes.
writeFlags	Controls the write command. Use 0 (zero) for this field.
pByteCount	A pointer to the returned number of bytes written. This value is NULL if no bytes were written.
pSampleIdx	A pointer to the index of the next sample in the stream. Specify NULL, if you do not need this value.
pTime	A pointer to the returned current time position within the stream, in ms. Specify NULL, if you do not need this value.

Return values

Return value	Description
SUCCESS	
MMERR_BUFFER_TOO_SMALL	No entire media sample found in ***pBuffer***. You need to provide a larger buffer.
MMERR_INCOMP_ACCESS_MODE	Stream was not opened for read operation.
MMERR_INTERNAL_RESOURCE	Error on internal resource.
MMERR_INVALID_STREAM	***pMMStream*** is invalid.
MMERR_INVALID_POINTER	***pBuffer*** is null.
MMERR_MEDIA_FORMAT	Error in media data format.

Details

The **mmWriteStream** function writes stream data to an intermediate holding area if the file that will contain the stream was opened with one of these settings:

- FORMAT_FLAG_WR_NORMAL
- FORMAT_FLAG_TEMP_MEM
- FORMAT_FLAG_TEMP_FILE

In this situation, the stream data is copied from the intermediate holding area to the file when you invoke **mmCloseFile**.

If the file was opened with the FORMAT_FLAG_WR_DIRECT setting, then the data is written to the media stream directly when you invoke **mmWriteStream**. For information about opening a file, see **mmOpenStream**.

Example

```
DWORD ret;

ret = mmWriteStream(&mmStream, pBuffer, wrSize, NULL, &byteCount, NULL, NULL);
if (ret != SUCCESS)
{
    printf("\nERROR: failed to write  the stream into the file. ret (%d)\n", ret);
}
```

10 Multimedia File Interface library structures

Using the MMFI library structure reference

This section provides a reference to the structures used by MMFI functions in the Video Messaging Server Interface. It includes structure definitions and field descriptions.

The following table lists the MMFI structures by function. Only those functions that have associated structures are included in the list.

MMFI function	Associated structures
mmGetFileInfo	FILE_INFO_DESC The following multimedia file presentation block structures: • FILE_INFO_BLOCK_HEADER • FILE_INFO_PRESENTATION The following multimedia file stream block structures: • FILE_INFO_BLOCK_HEADER • FILE_INFO_CODEC_AMR • FILE_INFO_CODEC_H263 • FILE_INFO_CODEC_H264 • FILE_INFO_CODEC_MPEG4 • FILE_INFO_FORMAT_AUDIO • FILE_INFO_FORMAT_VIDEO • FILE_INFO_STREAM_HEADER
mmGetVersion	MMFI_VERS_INFO
mmOpenFile	FILE_FORMAT_DESC
mmOpenStream	DATA_FORMAT_DESC DATA_FORMAT_INFO

DATA_FORMAT_DESC

Describes the format of a multimedia stream. This structure is used by **mmOpenStream** when it is invoked in write mode to configure the data format. It is used by **mmOpenStream**, when it is invoked in read mode to return the format of the existing file and configure the output data format (using the format and flags fields).

Definition

```
typedef struct
{
    WORD    format;
#define FORMAT_NMS_PACKETIZED 0
#define FORMAT_RAW 1
    DWORD   flags; //!< Format flags (bitmap).
#define FORMAT_FLAG_AMR_IF2          0x001
#define FORMAT_FLAG_AMR_DTX          0x002
#define FORMAT_FLAG_H263_2190        0x004
#define FORMAT_FLAG_RTP_HINTS        0x008
#define FORMAT_FLAG_RTP_SETTINGS     0x0010
#define FORMAT_FLAG_HEADER_CHECK     0x0020
#define FORMAT_FLAG_MEM_CHUNKS       0x0080
#define FORMAT_FLAG_DCI_INFO       0x0100
#define FORMAT_WRITE_SYNC_POINTS     0x2000
    WORD     rtpTimestampOffset;
    WORD     rtpSequenceOffset;
    WORD     rtpPayloadType;
#define DEFAULT_PAYLOAD_TYPE 0
    WORD     rtpPayloadSize;
#define DEFAULT_PAYLOAD_SIZE 0
    BYTE     videoProfile;
    BYTE     videoLevel;
    BYTE     *pdciInfo;
    int      dciInfoSize;
WORD      videoAggregationThreshold;
} DATA_FORMAT_DESC;
```

Fields used with all streams in all modes

Field name	Description
format	Format of the output data. Valid values are: • FORMAT_NMS_PACKETIZED - NMS native format (default). • FORMAT_RAW - Raw format. Uses bit stream or hint RTP packets.
flags	Bit mask that specifies format options for storing data. Valid values for both read mode and write mode are: • FORMAT_FLAG_AMR_IF2 - AMR format 2. The default is format 1. • FORMAT_FLAG_RTP_HINTS - In write mode, generates hint tracks. In read mode, if this flag is set and no hint tracks are present, no error is reported. The read operation will proceed and rely on the media tracks. Valid values for read mode only are: • FORMAT_FLAG_AMR_DTX - AMR DTX is turned on. The default is no DTX. • FORMAT_FLAG_H263_2190 - Use RFC2190 format H.263 packets. The default is RFC2429. • FORMAT_FLAG_RTP_SETTINGS - Force RTP settings. The default is do not force RTP settings. When you force RTP settings, there are additional fields you need to fill in, as described under "Fields used with media streams," below. Valid values for write mode only are: • FORMAT_FLAG_HEADER_CHECK 0x0020 - Check NMS Native headers. The default is not to check NMS Native headers. • FORMAT_FLAG_MEM_CHUNKS - Use multiple memory chunks to write a stream. The default is to use a single memory buffer. • FORMAT_FLAG_DCI_INFO - Use DCI information for MPEG-4 or H.264 video specified in the pdciInfo field. The default is not to use DCI information (ignore the pdciInfo field). • FORMAT_WRITE_SYNC_POINTS - Write synchronization points to the synchronization table. The synchronization table is used in random access operations.

Fields used with media streams (FORMAT_FLAG_RTP_SETTINGS = 1)

Field name	Description
rtpTimestampOffset	The timestamp of the first RTP packet.
rtpSequenceOffset	The sequence number of the first RTP packet.
rtpPayLoadType	The RTP payload type. Use DEFAULT_PAYLOAD_TYPE to use the default value, according to the codec type.
rtpPayLoadSize	The RTP payload maximum size. Use DEFAULT_PAYLOAD_SIZE to use the default value, according to the codec type.

Fields used when writing a video stream

Field name	Description
videoProfile	H.263 profile. Valid values are: • 0 = H.263 profile 0, level 10 - 30 • 3 = H.263+ profile 3, level 10 - 30
videoLevel	The codec standard level. Valid values are: • 10, 20, and 30 = H.263 or H.263+ level • 0 - 3 = MPEG-4 level • 1 - 1.2 = H.264 level
pdciInfo	DCI information for MPEG-4 or H.264.
dciInfoSize	Buffer size for DCI information in the pdciInfo field.
videoAggregationThreshold	Video aggregation threshold. Valid values are 0 or 1000 bytes to the maximum video size allowed in an RTP packet. 0 = Aggregation is not enabled. Not 0 = Video is aggregated such that there may be multiple video frames in a packet. **Note:** This parameter is for future use (not currently supported).

DATA_FORMAT_INFO

Provides size information for the opened stream. This structure is used by **mmOpenStream** in read mode.

Definition

```
typedef struct
{
    DWORD estTotalStreamSize;
    DWORD minReadBufferSize;
} DATA_FORMAT_INFO;
```

Fields

Field name	Description
estTotalStreamSize	Estimated total size of the whole stream data, including associated headers, in bytes. To read the whole stream with a single **mmReadStream** call, the application must provide a buffer greater than or equal to ***estTotalStreamSize.***
minReadBufferSize	Size of the largest sample in the stream, in bytes. To read the stream data, the application must provide a buffer greater than or equal to ***minReadBufferSize***.

FILE_FORMAT_DESC

Describes the format of the multimedia file to be written. This structure is used by **mmOpenFile** when it is invoked in write mode, to configure the file format.

Definition

```
typedef struct
{
    char     format [8];
    DWORD    version;
    DWORD    flags;
    DWORD    maxFileSize;
    DWORD    maxFileTime;
    DWORD    interleave;
} FILE_FORMAT_DESC;
```

Fields

Field name	Description
format [8]	File format string. For 3GP file format, this field specifies the major brand string. The default is 3GP6. A value of NULL indicates the default format.
version	Format version of the file. For 3GP files, the format is (x * 256 + y), which refers to TS 26.244 v6.x.y. The default is 256.
flags	Specifies how to store media data in the file when the **mmWriteStream** function is invoked. Valid values are: • FORMAT_FLAG_WR_NORMAL (default) - Store media data in temporary storage. • FORMAT_FLAG_WR_DIRECT -Store media data directly in the file. • FORMAT_FLAG_TEMP_MEM - Use allocated memory to store media data. • FORMAT_FLAG_TEMP_FILE - Store media data in intermediate files.
maxFileSize	Maximum file size, in bytes, for all media streams. A value of 0 (zero) indicates that the file size is not limited.
maxFileTime	Maximum file duration, in milliseconds, for all included media streams. A value of 0 (zero) indicates that the time is not limited.
interleave	Maximum interleave depth in presentation, in milliseconds. A value of 0 (zero) indicates that media streams are interleaved by 250 milliseconds.

FILE_INFO_DESC

Indicates the type of file described in the file info block. FILE_INFO_DESC is the descriptor for a multimedia file. This structure is used by **mmGetFileInfo**.

Definition

```
typedef struct
    {
    WORD  fileInfoSize;
    BYTE  *pFileInfo;
    WORD  fileType;
#define FILE_TYPE_UNKWN 0
#define FILE_TYPE_3GP 1
    char  format [8];
    DWORD version;
    DWORD flags;
#define DESC_FLAG_INSUFF_SPACE 1
    WORD  NbSyncPoints;
  WORD  sdpBuferSize
    WORD  reserved [4];
} FILE_INFO_DESC;
```

Fields

Field name	Description
fileInfoSize	Size of the file info block described by this descriptor.
pFileInfo	Pointer to the file info block.
fileType	Type of file. Valid values are: • FILE_TYPE_UNKWN • FILE_TYPE_3GP
format [8]	Specifies the file format string. For 3GP file format, this field specifies the major brand string. The default is 3GP6. A value of NULL indicates the default format.
version	Format version of the file. For 3GP files, the format is (x * 256 + y), which refers to TS 26.244 v6.x.y. The default is 256.
flags	Contains a value of DESC_FLAG_INSUFF_SPACE_1 if **mmGetFileInfo** fails to fill in the entire information structure.
NbSyncPoints	The number of sync points in the sync point table. If a sync point table was not created, the value will be 0.
sdpBufferSize	The total number of bytes for the SDP information in a 3GP file. If SDP is not stored in a 3GP file, the value is 0.
reserved	Reserved for future use.

MMFI_VERS_INFO

Contains information about the version and supported formats of the MMFI library. It is used by **mmGetVersion**.

Definition

```
typedef struct
{
    WORD infoSize;
    WORD majorRev;
    WORD minorRev;
    WORD build;
    WORD fileType [8];
} MMFI_VERS_INFO;
```

Fields

Field name	**Description**
infoSize	Number of bytes written to this information structure, including this size.
majorRev	Major revision number.
minorRev	Minor revision number.
build	Build revision number.
fileType	Types of files supported; terminated with a 0. Only 3GP files (FILE_TYPE_3GP) are supported.

Multimedia file presentation block structures

There are two structures in the presentation block of a multimedia file:

- FILE_INFO_BLOCK_HEADER
- FILE_INFO_PRESENTATION

These structures are returned by **mmGetFileInfo**.

FILE_INFO_BLOCK_HEADER

Identifies the block as a presentation block and contains the total size of the block.

Definition

```
typedef struct
{
    WORD blkSize;
    WORD blkType;
#define BLK_TYPE_PRESENTATION 1
#define BLK_TYPE_STREAM 2
} FILE_INFO_BLOCK_HEADER;
```

Fields

Field	Description
blkSize	Total size of the block in which this header is included, in bytes.
blkType	Identifies the block of information. Valid values are: • BLK_PRESENTATION - Block is a presentation block. • BLK_TYPE_STREAM - Block is either an audio stream block or a video stream block.

FILE_INFO_PRESENTATION

Contains presentation-level information.

Definition

```
typedef struct
{
FILE_INFO_BLOCK_HEADER blockHeader;
    DWORD  creationTime;
    DWORD  duration;
    WORD   streamCount;
    WORD   streams;
    DWORD  flags;
    DWORD  maxBitrate;
    DWORD  maxInterleave;
} FILE_INFO_PRESENTATION;
```

Fields

Field	Description
blkHeader	A pointer to the FILE_INFO_BLOCK_HEADER structure that identifies the block as a presentation block and contains the total size of the block.
creationTime	Creation date of the presentation, in seconds.
duration	Global play duration of the presentation, in seconds. Corresponds to the duration of the longest stream or track.
streamCount	Number of streams or tracks found in the file info block.
streams	Number of streams as indicated in presentation. The streams field is equal to 0 (zero), if this counter is not supported by the file format. This number can be different from the streamCount field.
flags	Reserved for future use.
maxBitrate	Reserved for future use.
maxInterleave	Maximum stream interleave depth in presentation, in milliseconds.

Multimedia file stream block structures

This topic describes the structures in the stream block of a multimedia file. A subset of these structures is returned by **mmGetFileInfo**.

This topic contains the following information:

- General structure of a stream block
- Level 1: Block header structure
- Level 2: Stream header structure
- Level 3: Media format structures
- Level 4: Codec-specific structures

General structure of a stream block

The following table shows the general structure of a stream block:

Block	Structure	Quantity	Description
blkHeader	FILE_INFO_BLOCK_HEADER	1	Block header, with type and size of the block.
header	FILE_INFO_STREAM_HEADER	1	Stream or track global information.
format	FILE_INFO_FORMAT_AUDIO or FILE_INFO_FORMAT_VIDEO	1	Audio or video stream format information.
codec	FILE_INFO_CODEC_AMR or union of FILE_INFO_CODEC_H263 or FILE_INFO_CODEC_MPEG4	0 or 1	Codec-specific information.

Level 1: Block header structure

Level 1 contains block header information that identifies the block as a stream block and contains the total size of the block. This information is contained in the FILE_INFO_BLOCK_HEADER structure.

Definition

```
typedef struct
{
    WORD blkSize;
    WORD blkType;
#define BLK_TYPE_PRESENTATION 1
#define BLK_TYPE_STREAM 2
} FILE_INFO_BLOCK_HEADER;
```

Fields

Field	Description
blkSize	Total size of the block in which this header is included, in bytes.
blkType	Identifies the block of information. BLK_STREAM identifies the block as a stream block.

Level 2: Stream header structure

Level 2 contains header information specific to a stream, including the type of codec used by the stream. The information is contained in the FILE_INFO_STREAM_HEADER structure.

Definition

```
typedef struct
{
    WORD blkSize;
    WORD streamType;
#define STREAM_TYPE_UNKNWN 0
#define STREAM_TYPE_AUDIO 1
#define STREAM_TYPE_VIDEO 2
    WORD codec;
#define S_CODEC_UNKNWN 0
#define S_CODEC_AMR 1
#define S_CODEC_AAC 2
#define S_CODEC_H263 3
#define S_CODEC_MPEG4 4
#define S_CODEC_AMR_WB 5
#define S_CODEC_AMR_WB_EXT 6
#define S_CODEC_H264 7
    DWORD streamID;
    DWORD creationTime;
    DWORD duration;
    DWORD rate;
    WORD altGroupID;
    WORD swiGroupID;
    WORD selectAttributes;
#define SELECT_ATTR_LANGUAGE 0x0001
#define SELECT_ATTR_BANDWIDTH 0x0002
#define SELECT_ATTR_CODEC 0x0004
#define SELECT_ATTR_SCREENSIZE 0x0008
#define SELECT_ATTR_MAXPKTSIZE 0x0010
#define SELECT_ATTR_STREAMTYPE 0x0020
    DWORD duration;
    DWORD rate;
    char handlerName [8];
    DWORD sampleSize;
#define SAMPLE_SIZE_VARIABLE 0
    DWORD sampleCount;
    DWORD maxSampleSize;
    DWORD byteCount;
    DWORD flags;
    DWORD startDelay;
} FILE_INFO_STREAM_HEADER;
```

Fields

Field	Description
blkSize	Total size of the FILE_INFO_STREAM_HEADER block, in bytes. The default value is 44.
streamType	Type of stream. Determines whether the stream block is an audio or video stream block. Valid values are: • STREAM_TYPE_UNKNWN - Ignore the rest of the stream block. • STREAM_TYPE_AUDIO - Stream block is a video block. • STREAM_TYPE_VIDEO - Stream block is an audio stream block.
codec	Type of codec. Determines whether the codec structure is an audio or video structure. Valid values are: • CODEC_UNKNWN - Ignore the rest of the stream block. • CODEC_AMR - Codec is AMR. • CODEC_H263 - Codec is H.263. • CODEC_MPEG4 - Codec is MPEG-4.
streamID	Identifier of the stream or track.
creationTime	Creation date of the stream or track.
altGroupID	Reserved for future use.
swiGroupID	Reserved for future use.
selectAttributes	Reserved for future use.
duration	Play duration of the stream or track.
rate	Sampling rate for audio stream (in Hz) or frame rate for video stream (in frames per second).
handlerName	Media handler name.
sampleSize	Size of the sample. The value is 0 (zero) if the size is variable.
sampleCount	Number of samples in the stream.
maxSampleSize	Size of the largest sample in the stream. The value is 0 (zero) if the information is not available.
byteCount	Total number of bytes in the stream or track.
flags	Reserved for future use.
startdelay	Reserved for future use.

Level 3: Media format structures

Level 3 contains the following media format structures:

- FILE_INFO_FORMAT_AUDIO
- FILE_INFO_FORMAT_VIDEO

The structure used depends on the codec specified in the stream header.

Note: When new fields are added, the blkSize field of blkHeader must reflect the new size of the block.

FILE_INFO_FORMAT_AUDIO

Contains audio-specific stream format information. It is always present in an audio stream block, and is common to all audio codecs.

Definition

```
typedef struct

    WORD blkSize;
    WORD channelCount;
    #define CHANNEL_COUNT_MONO 1
    #define CHANNEL_COUNT_STEREO 2
    #define CHANNEL_COUNT_51 6
    WORD sampleSize;
    WORD flags;
FILE_INFO_FORMAT_AUDIO;
```

Fields

Field	Description
blkSize	Total size of the FILE_INFO_FORMAT_AUDIO block, in bytes. The default value is 10.
channelCount	Number of audio channels: 1 = CHANNEL_COUNT_MONO 2 = CHANNEL_COUNT_STEREO 6 = CHANNEL_COUNT_51
sampleSize	Number of bits per audio sample.
flags	Reserved for future use.

FILE_INFO_FORMAT_VIDEO

Contains video-specific stream format information. It is always present in a video stream block, and is common to all video codecs.

Definition

```
typedef struct
{
    WORD blkSize;
    WORD width;
    WORD height;
    INT16 layer;
    WORD hResolution;
    WORD vResolution;
    DWORD flags;
} FILE_INFO_FORMAT_VIDEO;
```

Fields

Field	Description
blkSize	Total size of the FILE_INFO_FORMAT_VIDEO block, in bytes. The default value is 20.
width	Picture width, in pixels. For example, 176 for QCIF.
height	Picture height, in pixels. For example, 144 for QCIF.
layer	Front-to-back ordering. 0 is the normal view. Negative values are in front.
hResolution	Picture horizontal resolution, in pixels per inch. For example, 72.
vResolution	Picture vertical resolution in pixels per inch. For example, 72.
flags	Reserved for future use.

Level 4: Codec-specific structures

Level 4 contains the following codec information structures:

- FILE_INFO_CODEC_AMR
- FILE_INFO_CODEC_H263
- FILE_INFO_CODEC_MPEG4

The structure used depends on the codec specified in the stream header.

FILE_INFO_CODEC_AMR

Contains AMR codec information.

Definition

```
typedef struct
{
    WORD modeSet; /* bitmap of AMR modes possibly present in stream. b0=mode 0 etc. */
    #define S_AMR_MODE_475 0x0001
    #define S_AMR_MODE_515 0x0002
    #define S_AMR_MODE_590 0x0004
    #define S_AMR_MODE_670 0x0008
    #define S_AMR_MODE_740 0x0010
    #define S_AMR_MODE_795 0x0020
    #define S_AMR_MODE_102 0x0040
    #define S_AMR_MODE_122 0x0080
    #define S_AMR_MODE_SID 0x0100
    #define S_AMR_MODE_NODATA 0x8000
    #define S_AMR_MODE_ALL 0x81FF
    BYTE modeChangePeriod;
    BYTE framesPerSample;
    char vendor [8];
    // new fields may be added here in future release
} FILE_INFO_CODEC_AMR;
```

Fields

Field	Description
modeSet	Bitmap of AMR modes that can be present in the stream. Valid values are: • AMR_MODE_475 - Mode 0, 4.75 kbit/s. • AMR_MODE_515 - Mode 1, 5.15 kbit/s. • AMR_MODE_590 - Mode 2 - 5.90 kbit/s. • AMR_MODE_670 - Mode 3, 6.70 kbit/s. • AMR_MODE_740 - Mode 4, 7.40 kbit/s. • AMR_MODE_795 - Mode 5, 7.95 kbit/s. • AMR_MODE_102 - Mode 6, 10.2 kbit/s. • AMR_MODE_122 - Mode 7, 12.2 kbit/s. • AMR_MODE_SID- Mode 8, AMR SID. • AMR_MODE_NODATA - No data. • AMR_MODE_ALL - All modes may be present.
modeChangePeriod	A value of 0 (zero) specifies no restriction.

Field	Description
framesPerSample	Number of frames per sample.
vendor	Four-character string representing the codec's manufacturer.

FILE_INFO_CODEC_H263

Contains H.263 codec information.

Definition

```
typedef struct
{
    BYTE level;
    BYTE profile;
    DWORD avgBitrate;
    DWORD maxBitrate;
    char vendor [8];
    // new fields may be added here in future release
} FILE_INFO_CODEC_H263;
```

Fields

Field	Description
level	H.263 standard level. Valid values are 10, 20, and 30.
profile	H.263 profile. The only valid value is 0.
avgBitrate	Reserved for future use.
maxBitrate	Reserved for future use.
vendor	Reserved for future use.

FILE_INFO_CODEC_H264

Contains H.264 codec information.

Definition

```
typedef struct
{
WORD  dcStreamType;
WORD  dcSpecInfoSize;
#define MMFI_MAX_H264_DCI_SZ 1024
BYTE  pDcSpecInfo[MMFI_MAX_H264_DCI_SZ];
WORD  dcSpecInfoSize3gppFormat;
BYTE  pDcSpecInfo3gppFormat[MMFI_MAX_H264_DCI_SZ];
BYTE  level;
} FILE_INFO_CODEC_H264;
```

Fields

Field	Description
dcStreamType	streamType field of the MPEG-4 DecoderConfig descriptor.
dcSpecInfoSize	Size of decoder specific information, in bytes.
pDcSpecInfo	Points to specific information found in the MPEG-4 DecoderConfig descriptor.
dcSpecInfoSize3gppFormat	Size of decoder specific information, 3gpp format, in bytes. The 3gpp format version contains H.264 start codes.

Field	**Description**
pDcSpecInfo3gppFormat	Points to specific information found in the MPEG-4 DecoderConfig descriptor, 3gpp format.
level	H.264 level.

FILE_INFO_CODEC_MPEG4

Contains MPEG-4 codec information.

Definition

```
typedef struct
{
    WORD  dcStreamType;
    WORD  dcBufferSize;
    DWORD dcAvgBitrate;
    DWORD dcMaxBitrate;
    WORD  dcSpecInfoSize;
#define MMFI_MAX_MPEG4_DCI_SZ 128
    BYTE pDcSpecInfo[MMFI_MAX_MPEG4_DCI_SZ];

BYTE level;
} FILE_INFO_CODEC_MPEG4;
```

Fields

Field	**Description**
dcStreamType	streamType field of the MPEG-4 DecoderConfig descriptor.
dcBufferSize	Maximum size of sample, in bytes, as indicated in the MPEG-4 DecoderConfig descriptor. For example, minimum size of the decoder buffer.
dcAvgBitrate	Average bitrate, as indicated in the MPEG-4 DecoderConfig descriptor.
dcMaxBitrate	Maximum bitrate, as indicated in the MPEG-4 DecoderConfig descriptor.
dcSpecInfoSize	Size of decoder specific information, in bytes.
pDcSpecInfo	Points to specific information found in the MPEG-4 DecoderConfig descriptor. If present, this data is stored at the end of the current stream block.
level	MPEG-4 level.

Index