3G-324M Interface Developer's Reference Manual

9000-62471-19

NMS COMMUNICATIONS

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

Every effort has been made to ensure the accuracy of this manual. However, due to the ongoing improvements and revisions to our products, NMS Communications cannot guarantee the accuracy of the printed material after the date of publication or accept responsibility for errors or omissions. Revised manuals and update sheets may be published when deemed necessary by NMS Communications.

P/N 9000-62471-19

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	MVH, 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	October 2007	DEH, Video Access 3.1
1.9	February 2009	DEH, Video Access 3.2
Last modified: January 19, 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 *3G-324M Interface Developer's Reference Manual* is one in a set of manuals that describe the Video Access product. It describes how to use the 3G-324M Interface to connect with 3G-324M terminals capable of audio and video. It also describes the 3G-324M Interface capabilities and functions.

This manual targets video application developers who use Natural Access. This document defines telephony terms where applicable, but assumes that you are familiar with the 3G-324M standard, 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 3G-324M Interface

3G-324M Interface overview

The 3G-324M Interface provides a flexible API for bridging 3G-324M clients into an IP network to provide access to various enhanced services applications. It allows for simple and flexible call setup with 3G-324M terminals on one side of the interface, while providing control for IPv4 or IPv6 media endpoints on the other side. Data must enter the interface in 3G-324M format.

For information about the 3G-324M 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.

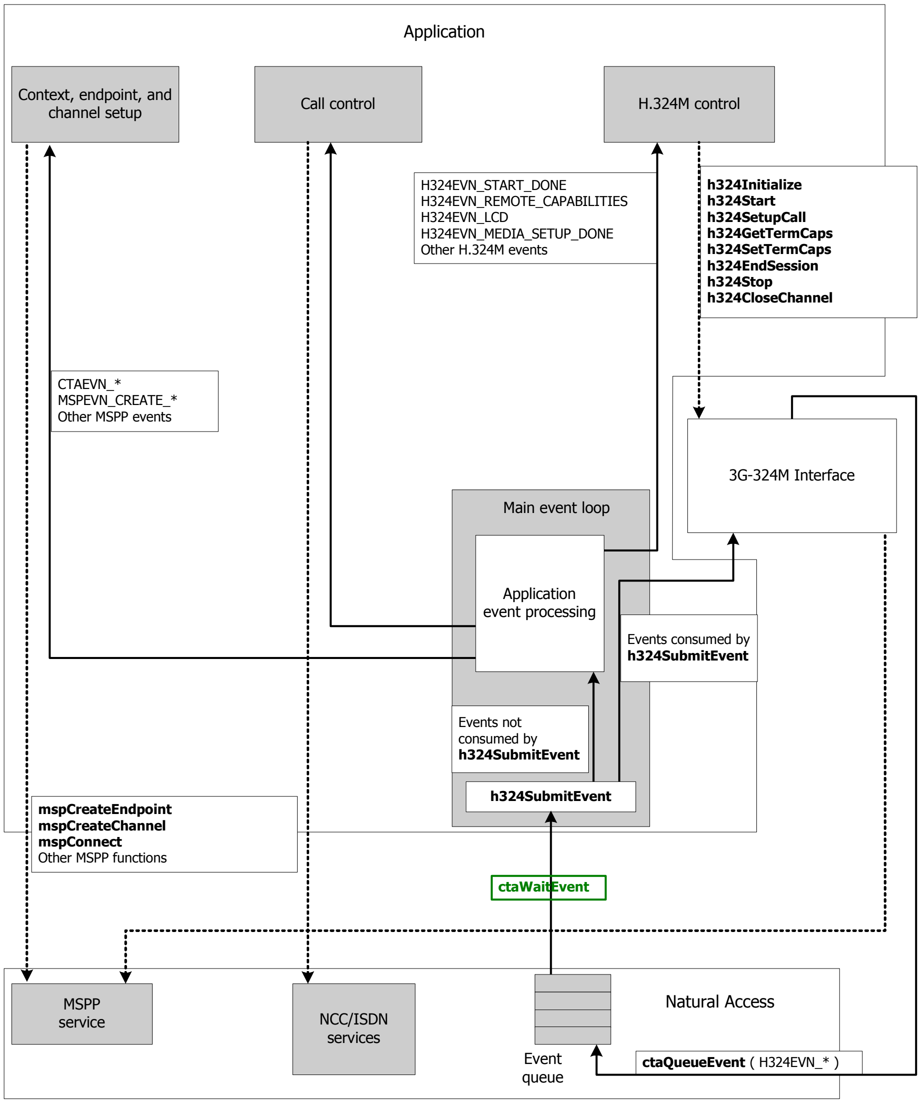
Note: For an additional layer of detail about Video Access structures, refer to the Video Access header files.

3 Configuring the 3G-324M Interface

Understanding the host application model

The host application can use any supported NMS programming model (single-threaded programming, multi-threaded programming, or multi-process programming). All NMS-supplied APIs are fully thread-safe. For information about Natural Access programming models, refer to the *Natural Access Developer's Reference Manual*.

The following illustration shows the relationship between the application and the 3G-324M Interface:



Designing the 3G-324M Interface channel configurations

Design the 3G-324M Interface channel configurations based on the media codecs selected and whether audio transcoding is required.

There are two types of configurations you can set up:

- A pass-through configuration, when no audio transcoding is required.
- An audio transcoding configuration, which is required when audio transcoding to G.711 or another supported Fusion codec is to be performed.

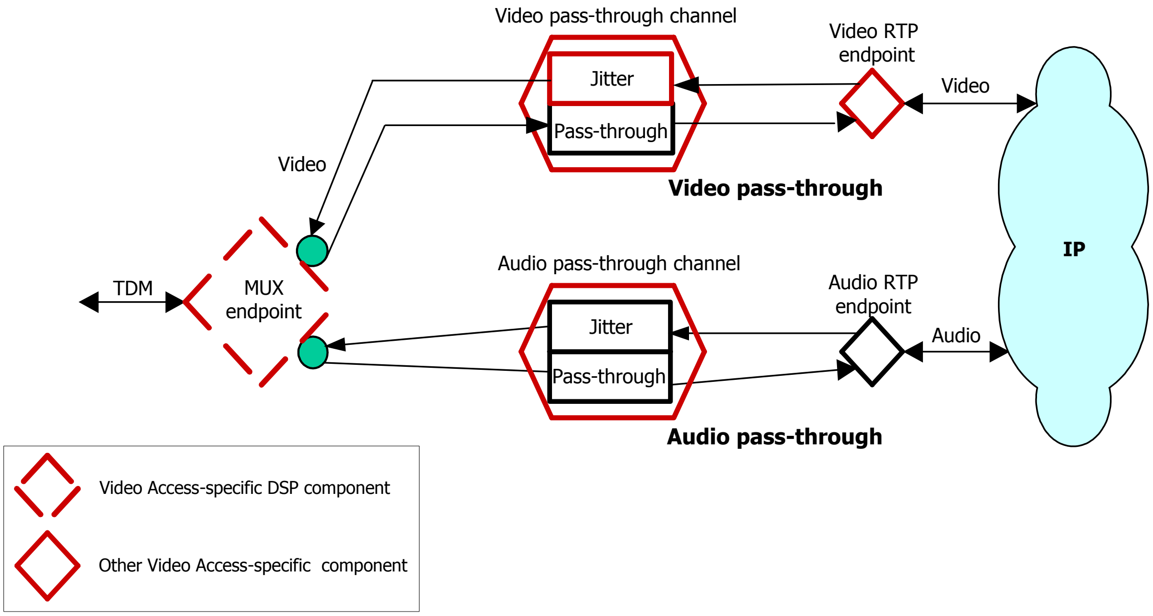
Pass-through configuration

If audio transcoding is not required, the application must create the following for each 3G-324M port:

- A MUX/DEMUX endpoint, associated with a PSTN timeslot. The remainder of this manual refers to the MUX/DEMUX endpoint as a MUX endpoint.
- A video pass-through channel and a video RTP endpoint (MPEG-4, H.263, or H.264).
- An audio pass-through channel and a Fusion audio RTP endpoint.

The video channel must connect with the MUX endpoint and the video RTP endpoint. The audio channel must connect with the MUX endpoint and the audio RTP endpoint. Both pass-through channels must then be enabled.

The following illustration shows a pass-through full-duplex configuration (simplex configurations can also be used):

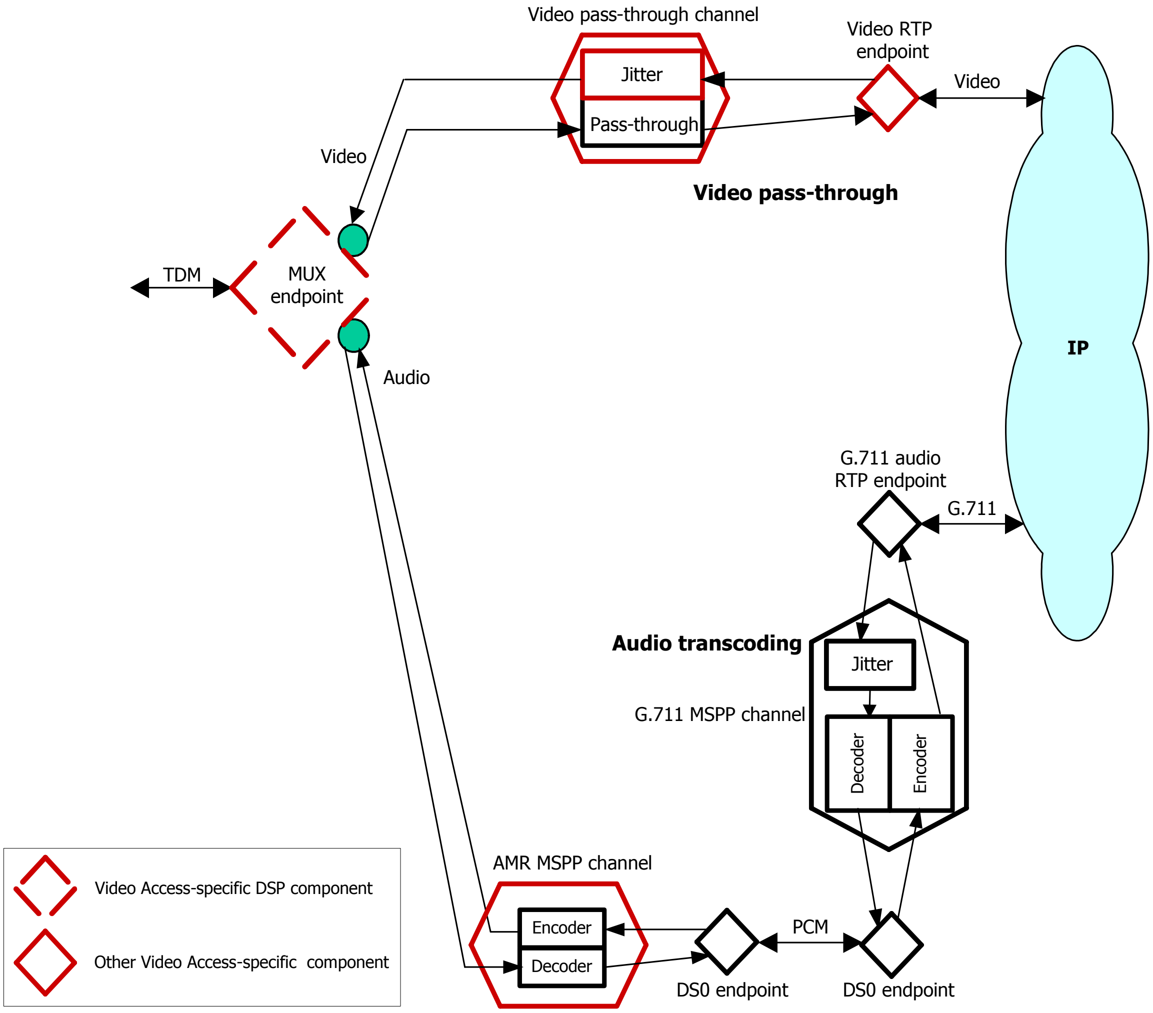


Audio transcoding configuration

If the application requires G.711 encoded RTP packetized audio on the IP side, then you must do the following:

- Select the AMR audio codec on the 3G-324M side.
- Create an AMR MSPP channel and a G.711 MSPP channel between the respective sets of endpoints, as illustrated in the diagram below.
- Use TDM switching to connect the two DS0 endpoints. This completes the path between the MUX endpoint and the RTP endpoint.
- Connect the AMR MSPP transcoding channel to the Demux prior to starting the 3G-324M negotiation.

The following illustration shows a full-duplex audio transcoding configuration. (Simplex configurations can also be used.)



MUX and DEMUX capabilities

The MUX and DEMUX (DPF) data processing DSP function implements a selective subset of the capabilities described in the ITU H.223 specification and H.324 specification for CCSRL/NSRP.

The following table summarizes the multiplexer-supported features:

Specification	**Features**	**Supported**	**Not supported**
H.223	Multiplexer	X	
	AL1 protocol	X	
	AL2 protocol	X	
	AL3 protocol (no re-transmission)	X	
H.223 Annex A	Multiplexer double flag Mode		X
H.223 Annex B	Golay coding for multiplexer header stuffing PM synchronization	X	
H.223 Annex C	Additional error reduction capabilities		X
H.223 Annex D	Additional error reduction capabilities		X
H.324 Annex A, Annex C	CCSRL/NSRP/WNSRP	X	
H.324 Annex K	MONA	X	

Supported audio and video formats

Applications must use a single audio and video codec for the duration of each video call.

The following table presents the audio and video formats that are supported on both sides of the gateway:

Type	Format
PSTN audio	AMR-NB G.723-1
PSTN video	H.263 baseline level 10 H.263+ profile 3 level 10 H.264 baseline profile level 1 ISO/IEC 14496-2 MPEG-4 simple profile level 0
IP audio	AMR-NB over RFC 3267, as described below. G.723-1 G.711 Conformance with RFC 3267 is maintained in the following manner: • Supported: Octet-aligned mode • Not supported: Received Codec Mode Requests (CMR), forward error correction, interleaving, robust sorting, UEP/UED bit error detection schemes, multi-channel payloads
IP video	H.263 Baseline level 10 over RFC 2190 and RFC 2429 H.263+ profile 3 level 10 over RFC 2429 H.264 Baseline profile level 1 over RFC 3984, no interleaving MPEG-4 simple profile level 0 over RFC 3016

The 3G-324M Interface supports pass-through (no transcoding) of the following codecs:

- AMR narrow band audio, 3GPP TS 26.090 version 5.3, compliant with RFC 3267 for RTP packetization
- G.723.1 audio
- MPEG-4 simple profile level 0 video, compliant with RFC 3016 for RTP packetization
- H.263 baseline video, as specified in annex X (level 10), compliant with RFC 2429 and RFC 2190 for RTP packetization, with the exception of transmit support in mode B in RFC 2190
- H.263+ profile 3 level 10, compliant with RFC 2429 for RTP packetization
- H.264 baseline profile level 1 video, compliant with RFC 3984 for RTP packetization, without interleaving

Pass-through means that encoded frames received from the PSTN are H.223 demultiplexed, encapsulated in RTP-packets, and output over IP. No transcoding is performed on the frame payloads.

For data transferred from IP to PSTN, a jitter buffer function is provided on both audio and video streams.

When receiving the AMR NB codec from the PSTN interface, the audio logical channel of the H.223 MUX can optionally be transcoded to G.711 for transmission over IP.

Note: The application can convert AMR audio to codecs other than RTP G.711, at the expense of port density. Refer to CG board port densities in the *readme-va.txt* for more information. Refer to the *Fusion Developer's Manual* for a list of supported audio codecs on IP.

Tasks for configuring the 3G-324M Interface

To configure the 3G-324M Interface, the host application must perform the following tasks:

Task	Action	Use	For more information, see...
1	Configure the CG board.	Natural Access	*Configuring the board* on page 17.
2	Initialize Natural Access and open the following services: • MSPP • SWI • ISDN or SS7	Natural Access	*Natural Access Developer's Reference Manual.*
3	Initialize the H.324M Middleware.	3G-324M Interface	*Enabling fast call setup* on page 23 and *Initializing the H.324M Middleware* on page 25.
4	Create Natural Access contexts and event queues.	Natural Access	*Natural Access Developer's Reference Manual*.
5	Create the MUX endpoint.	3G-324M Interface	*Creating the MUX endpoint* on page 29.
6	Make switching connections from the trunk to the MUX endpoint, if needed.	Switching service	*Switching Service Developer's Reference Manual*.
7	Set up PSTN call control (ISDN, SS7).	PSTN service	*NMS ISDN for Natural Call Control Developer's Manual or* the *TDM for SS7 Developer's Reference Manual.*
8	Set up a 3G-324M session.	3G-324M Interface	For more information, see *Setting up a 3G-324M session* on page 29.
9	Create and connect MSPP endpoints and channels.	MSPP service with video-enhanced components	• *Creating an endpoint* on page 31. • *Creating a channel* on page 34. • *Connecting endpoints and channels* on page 36.
10	Set up tracing and monitoring functionality.	3G-324M Interface	*Setting up tracing and monitoring functionality* on page 36.
11	Stop a call or session.	3G-324M Interface	*Stopping a call or session* on page 37.

Configuring the board

This topic provides sample files that configure CG boards for a 3G-324M gateway application.

Note: When configuring a CG board for a video application, you must configure MUX DSP pools.

Refer to the CG board installation and developer's manual for general information about configuring the board.

Sample configuration file for the CG 6565 board

In this example, 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**).

The video-specific settings are shown in bold.

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

Sample configuration file for the CG 6060 board

In this example, 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 ten MUX/DEMUX ports, available on timeslots 0 through 59 (specified in **ctaOpenServices**).

The video-specific settings are shown in bold.

```
################################################################
# 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 = 10.118.7.121
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 = YESIPv6.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]  = 0xB
Hdlc[0..3].Boot              = YES
Hdlc[0..3].Comet.TxTimeSlot  = 16
Hdlc[0..3].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..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)
################################################################
# DOWNLOADABLE RUNTIME MODULES
DLMFiles[0]        = cg6060fusion
DLMFiles[1]        = c6060igen
################################################################
# DEBUG STUFF
DebugMask                   = 0x00
##############################################################
```

Sample configuration file for the CG 6000 board

In this example, 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**).

The video-specific settings are shown in bold.

```
################################################################
# 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
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)
################################################################
# DOWNLOADABLE RUNTIME MODULES
DLMFiles[0]                 = cg6krun
DLMFiles[1]                 = cg6kfusion
DLMFiles[2]                 = isdngen
################################################################
################################################################
# DEBUG STUFF
DebugMask                   = 0x00
##############################################################
```

Enabling fast call setup

The 3G-324M Interface supports the following techniques for speeding up 3G-324M call setup time:

- Packed H.245 messages
- WNSRP
- MONA

Note: You must obtain the appropriate license to use one of these techniques. For information, see the readme file for this release.

Packed H.245 messages

Packed H.245 messages allow grouping independent H.245 messages together into a single NSRP command frame. This reduces the number of message round-trips, and thus reduces call setup time. NMS optimizes this technique to achieve smart grouping of messages.

You can enable the packing of H.245 messages by setting the pack_h245 parameter to 1 in the H.324 Middleware configuration file. For information, see *Initializing the H.324M Middleware* on page 25.

WNSRP

WNSRP (Windowed Simple Retransmission protocol) is an H.245 transport improvement technique that is standardized in ITU-T Recommendation H.324 and accepted into the 3G-324M standard by 3GPP. WNSRP introduces a transmission window for the WNSRP messages sent by a terminal. The transmission of WNSRP H.245 messages allows for multiple independent messages to be sent without acknowledgement, which results in a more efficient use of bandwidth for each message.

The 3G-324M Interface supports WNSRP. The Interface falls back to regular NSRP for compatibility with non-WNSRP terminals.

Enable WNSRP by setting the nsrp_mode parameter to 3 in the H.324 Middleware configuration file. For information, see *Initializing the H.324M Middleware* on page 25.

MONA

MONA (media oriented negotiation acceleration), formally known as ITU-T Recommendation H.324 Annex K, was approved by the ITU-T in August 2006, and is recommended in 3GPP Release 7 in TR 26.911. MONA unites the technologies for H.324 call setup acceleration under a common framework.

The following table describes the MONA capabilities that the 3G-324M Interface supports:

Supported MONA capability	Description
Exchange of fast call setup capabilities and preferences for MPC MONA call setup	The 3G-324M Interface supports the use of MONA Media Pre-Configured Channels (MPC) to achieve the fast setup of media sessions. When multiple video codecs are supported by the application, only the preferred codec is used for the MONA MPC fast call setup attempt. The preferred codec is the first codec passed by the application to the H.324M Middleware. MONA MPC relies on specific codec configurations, so that the codec configuration does not have to be negotiated. When encoding video, use the configurations defined by MONA. For information, see ITU-T H.324 Annex K.
Fallback to MONA ACP technique	If the MONA MPC negotiation fails, the 3G-324M Interface uses accelerated H.245 signaling (ACP) as a fallback negotiation technique.
Fallback to regular call setup against legacy terminals	The 3G-324M Interface is fully compatible with legacy 3G-324M terminals.

You can enable MONA by setting the mona parameter to 1 in the H.324 Middleware configuration file. As recommended by 3GPP, always enable WNSRP when you enable MONA. For information about enabling MONA and WNSRP, see *Initializing the H.324M Middleware* on page 25.

Not supported

The 3G-324M Interface does not support:

- The Signaling Pre-Configured Channel (SPC) MONA technique.
- Pre-configured Channel Media frames encapsulated in MONA signaling Preference Messages.

Initializing the H.324M Middleware

Initialize the H.324M Middleware by using **h324Initialize**. This function must be called only once by the application.

The H.324M Middleware supports an optional *h324.cfg* configuration file that **h324Initialize** reads when invoked. The *h.324.cfg* file is located in the following directory:

- Windows: *c:\nms\vaccess\cfg\h324.cfg*
- Linux/Solaris: */opt/nms/vaccess/cfg/cfg/h324.cfg*

Each line of the configuration file specifies a different parameter value. The format of a line is:

```
Parameter = Value
```

Any text appearing after a # character is ignored as a comment. The following table describes the *h324.cfg* configuration file configuration parameters.

Parameter	Type	Default	Description
AsymmetricRecoveryMasterTimeout	int	4000	Amount of time, in ms, that the H.324M Middleware waits before starting the non-standard recovery procedure. This field applies only when AsymmetricVideoCodecsRecovery is set to 1 (enabled).
AsymmetricVideoCodecsRecovery	int	0	Enable or disable the non-standard recovery procedure when asymmetric video codecs are used. Values are: 0 – Disable the non-standard recovery procedure. 1 - Enable the non-standard recovery procedure. The H.324M Middleware can use the non-standard recovery procedure to recover a call when a 3G phone sends a unidirectional OLC request that uses an incorrect video codec (one that is not preferred by the master).
h245MsgLogDuration	int	0	If the h245 logging file is enabled, specifies the number of seconds to wait before changing to a new file. Default value 0, specifies unlimited duration.
h245MsgLogEnabled	int	0	Enable or disable logging decoded H.245 messages. Values are: 0 – Disable logging decoded H.245 messages 1 – Enable logging decoded H.245 messages

Parameter	Type	Default	Description
h245MsgLogNumFiles	int	1	Only used if the h245 logging file is enabled and h245TraceDuration does not equal 0. Specifies the maximum number of h245 log files to keep on the system. Valid range is 1 - 1000.
h245MsgLogFileName	string	*h245_msg.log.txt*	Log file name for decoded H.245 messages.
h245TraceDuration	int	0	If the h245 tracing file is enabled, specifies the number of seconds to wait before changing to a new file. A value of 0 (the default) specifies unlimited duration.
h245TraceNumFiles	int	1	Only used if the h245 tracing file is enabled and h245TraceDuration does not equal 0. Specifies the maximum number of 245 trace files to keep on the system. Valid range is 1 - 1000.
h245TraceFileName	string	*H245.log*	Trace file for logging H.245 protocol messages.
h245TraceLevel	int	0	Level of trace messages to generate. Bit mask values are: 0 - Disable trace levels. 1 – Generate H.245 error messages. 2 – Generate low-level H.245 messages. 4 – Generate high-level H.245 messages. 8 – Generate low-level H.245 debug messages. 16 – Generate high-level H.245 debug messages.
h245TraceMode	int	2	Where to write trace messages: 1 – Write trace messages to the console. 2 – Write trace messages to the trace file. 3 – Write trace messages to both the console and the trace file.

Parameter	Type	Default	Description
h245TraceModules	int	0	Category of messages to trace. Bit mask values are: 0 – Disable trace modules. 1 – Trace MSD messages. 2 – Trace TCS messages. 4 – Trace MES messages. 8 – Trace RME messages. 16 – Trace RTD messages. 32 – Trace OLC messages. 64 – Trace system messages. 128 – Trace request messages. 256 – Trace indication messages. 512 – Trace command messages. 1024 – ML messages. 2048 – Trace timer messages. 4096 – Trace memory messages.
h324TraceDuration	int	0	If the h324 tracing file is enabled, specifies the number of seconds to wait before changing to a new file. Default value specifies an unlimited duration.
h324TraceNumFiles	int	1	Only used if the h324 tracing file is enabled and h324TraceDuration does not equal 0. Specifies the maximum number of h324 trace files to keep on the system. Valid range is 1 - 1000.
LegacySyncFlagThreshold	int	5	Only used if MONA is not enabled. Specifies the number of H.223 Level 2 sync flags needed to return H324EVN_START_DONE. Valid range is 1 – 20.
max_n400_retrans_counter	int	5	N400 NSRP/WNSRP retransmission counter.
max_n402_counter	int	2	Only used if WNSRP is enabled. The H.324M Middleware switches to NSRP mode when it receives max_n402_counter NSRP responses.
maxAL2SDUSize	int	1024	Maximum AL2 SDU size in the terminal capability request.
maxAL3SDUSize	int	1024	Maximum AL3 SDU size in the terminal capability request.
mona	int	0	Enable or disable MONA. Values are: 0 – Disable MONA 1 – Enable MONA

Parameter	Type	Default	Description
nsrp_mode	int	2	NRSP protocol implementation. Values are: 2 – Enables NSRP and disables WNSRP. 3 – Enables NSRP and WNSRP.
pack_h245	int	2	Enable or disable the use of packed H.245 messages. Values are: 1 – Enable packed H.245 messages. 2 – Disable packed H.245 messages.
StartTimeoutDuration	int	5000	Amount of time (in ms) allowed for h324Start to complete processing before the Middleware generates an H324EVN_START_TIMER_EXPIRED event.
t401_timer_duration	int	1000	Duration, in ms, of the T401 NRSP/WNSRP acknowledgement timer.

Creating the MUX endpoint

Create the MUX endpoint for the 3G-324M Interface by using **mspCreateEndpoint** with the MSP_ENDPOINT_MUX identifier. This identifier is used in the eEpType field in the MUX_ENDPOINT_ADDR structure and in the eParmType field in the MUX_ENDPOINT_PARMS structure.

The following example shows how to declare that an endpoint is a MUX endpoint and how to set its timeslot:

```
GwConfig[i].MuxEp.Addr.eEpType          = MSP_ENDPOINT_MUX;
GwConfig[i].MuxEp.Addr.EP.Mux.nTimeslot = 5;
GwConfig[i].MuxEp.Param.eParmType       = MSP_ENDPOINT_MUX;
ret = mspCreateEndpoint(GwConfig[i].MuxEp.ctahd,
                        &GwConfig[i].MuxEp.Addr,
                        &GwConfig[i].MuxEp.Param,
                        &GwConfig[i].MuxEp.hd);
```

Note: The application does not need to interpret MUX unsolicited events, because these events are consumed by the H.324M Middleware. For more information, see *h324SubmitEvent* on page 129.

Setting up a 3G-324M session

Setting up a 3G-324M session involves the following:

- Starting the 3G-324M session
- Establishing terminal capabilities
- Handling events

Starting the 3G-324M session

Use **h324Start** to start the 3G-324M session. This function does the following:

- Creates an H.245 stack for the specified MUX.
- Sets the initial values that are needed to begin an exchange with the client (such as resetting PMSYNC and the video sequence number).

Establishing terminal capabilities

Use the following functions to establish terminal capabilities for the 3G-324M session:

Function	Description
h324SetupCall	Informs the H.324M Middleware that it can begin H.245 negotiations with the remote terminal.
h324GetTermCaps	Queries the H.324M Middleware for the 3G-324M Interface or remote terminal 3G-324M audio, video, and multiplexing capabilities.
h324SetTermCaps	Sets the local terminal capabilities and initiates the transfer of terminal capabilities (H.245 terminal capabilities exchange).

After you complete call setup, the Middleware returns one of the following events:

- H324EVN_CALL_SETUP_FAILED, if the call setup was not successful.
- H324EVN_MEDIA_SETUP_DONE, if the media was set up correctly for both audio and video.
- (MONA fast call setup only) H324EVN_FAST_CALL_SETUP_DONE notification event.

The Middleware also returns H324EVN_LCD in the following cases:

- The Middleware either accepted a unidirectional remote OLC request, or received a confirmation in the case of a bi-directional remote OLC.
- The remote terminal confirmed a bi-directional OLC request that was accepted by the Middleware.
- The Middleware received an accept response from the remote terminal.

Handling events

The following table describes how to handle events in the 3G-324M session:

Step	Action
1	Wait for Natural Access events by invoking **ctaWaitEvent**.
2	Submit all events received from Natural Access to the H.324M Middleware by invoking **h324SubmitEvent**. The Middleware processes these events, and returns them to the application when relevant.
3	Free buffers for all MSPP events that include buffers, except for those consumed by **h324SubmitEvent**.

Creating an endpoint

Create endpoints for the 3G-324M Interface by using **mspCreateEndpoint**. The 3G-324M Interface supports the following endpoint types:

- IPv4 audio
- IPv4 video
- IPv6 audio
- IPv6 video
- MUX

IPv4 audio endpoint types

The following table lists the IPv4 audio endpoint types most likely to be used in the 3G-324M Interface, along with their associated identifiers and initialization structures.

Note: G.723 audio is used for G.723 bypass, AMR audio is used for AMR bypass, and G.711 audio is used for AMR transcoding.

Endpoint type	Identifier	Initialization structure
G.723 audio, AMR audio, or G.711 audio full-duplex	MSP_ENDPOINT_RTPFDX	RTPRTCP_ENDPOINT_ADDR RTPRTCP_ENDPOINT_PARMS
G.723 audio, AMR audio, or G.711 audio simplex receive	MSP_ENDPOINT_RTPIN	RTPRTCP_ENDPOINT_ADDR RTPRTCP_ENDPOINT_PARMS
G.723 audio, AMR audio, or G.711 audio simplex send	MSP_ENDPOINT_RTPOUT	RTPRTCP_ENDPOINT_ADDR RTPRTCP_ENDPOINT_PARMS
DS0 (used for AMR transcoding)	MSP_ENDPOINT_DS0	DS0_ENDPOINT_ADDR DS0_ENDPOINT_PARMS

IPv4 video endpoint types

The following table lists the IPv4 video endpoint types that can be used in the 3G-324M Interface, along with their associated identifiers and initialization structures. The IPv4 video endpoint types use RTPRTCP_ENDPOINT_ADDR and RTPRTCP_ENDPOINT_PARMS as initialization structures.

Note: The endpoints listed in this table are Video Access-specific.

Endpoint type	Identifier
MPEG-4 full-duplex	MSP_ENDPOINT_RTPFDX_VIDEO_MPEG4
MPEG-4 simplex receive	MSP_ENDPOINT_RTPIN_VIDEO_MPEG4
MPEG-4 simplex send	MSP_ENDPOINT_RTPOUT_VIDEO_MPEG4
H.263 full-duplex	MSP_ENDPOINT_RTPFDX_VIDEO_H263
H.263 simplex receive	MSP_ENDPOINT_RTPIN_VIDEO_H263
H.263 simplex send	MSP_ENDPOINT_RTPOUT_VIDEO_H263
H.264 full-duplex	MSP_ENDPOINT_RTPFDX_VIDEO_H264
H.264 simplex receive	MSP_ENDPOINT_RTPIN_VIDEO_H264
H.264 simplex send	MSP_ENDPOINT_RTPOUT_VIDEO_H264

IPv6 audio endpoint types

The following table lists the IPv6 audio endpoint types most likely to be used in the 3G-324M Interface, along with their associated identifiers. All of these IPv6 audio endpoint types use RTPRTCP_V6_ENDPOINT_ADDR and RTPRTCP_V6_ENDPOINT_PARMS as initialization structures.

Endpoint type	Identifier
G.723 audio, AMR audio, or G.711 audio full-duplex for IPv6	MSP_ENDPOINT_RTPFDX_V6
G.723 audio, AMR audio, or G.711 audio simplex receive for IPv6	MSP_ENDPOINT_RTPIN_V6
G.723 audio, AMR audio, or G.711 audio simplex send for IPv6	MSP_ENDPOINT_RTPOUT_V6

IPv6 video endpoint types

Note: The endpoints listed in this table are Video Access-specific.

The following table lists the IPv6 video endpoint types that can be used in the 3G-324M Interface, along with their associated identifiers. The IPv6 video endpoint types use RTPRTCP_V6_ENDPOINT_ADDR and RTPRTCP_V6_ENDPOINT_PARMS as initialization structures.

Endpoint type	Identifier
MPEG-4 full-duplex for IPv6	MSP_ENDPOINT_RTPFDX_VIDEO_MPEG4_V6
MPEG-4 simplex receive for IPv6	MSP_ENDPOINT_RTPIN_VIDEO_MPEG4_V6
MPEG-4 simplex send for IPv6	MSP_ENDPOINT_RTPOUT_VIDEO_MPEG4_V6
H.263 full-duplex for IPv6	MSP_ENDPOINT_RTPFDX_VIDEO_H263_V6
H.263 simplex receive for IPv6	MSP_ENDPOINT_RTPIN_VIDEO_H263_V6
H.263 simplex send for IPv6	MSP_ENDPOINT_RTPOUT_VIDEO_H263_V6
H.264 full-duplex for IPv6	MSP_ENDPOINT_RTPFDX_VIDEO_H264_V6
H.264 simplex receive for IPv6	MSP_ENDPOINT_RTPIN_VIDEO_H264_V6
H.264 simplex send for IPv6	MSP_ENDPOINT_RTPOUT_VIDEO_H264_V6

MUX endpoint type

The following table lists the MUX endpoint type, along with its identifier and initialization structure.

Note: This endpoint is Video Access-specific.

Endpoint type	Identifier	Initialization structure
H.223 MUX/DEMUX	MSP_ENDPOINT_MUX	MUX_ENDPOINT_ADDR

Example

Create all endpoints using **mspCreateEndpoint**. For example:

```
ret = mspCreateEndpoint(GwConfig[i].MuxEp.ctahd,
&GwConfig[i].MuxEp.Addr,
&GwConfig[i].MuxEp.Param,
&GwConfig[i].MuxEp.hd);
```

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

Creating a channel

Create a channel for the 3G-324M Interface by using **mspCreateChannel**. The video and audio channel types that you can create for the Interface are described in the following table. All of the channel types use the MSP_CHANNEL_ADDR and the MSP_CHANNEL_PARAMETER initialization structures.

Video channel types

The following table lists the video channel types that can be used in the 3G-324M Interface, along with their associated identifiers. All of these channel types and identifiers are video-specific.

Channel type	Identifier
Video MPEG-4 full-duplex	MGVideoChannel
Video MPEG-4 simplex receive	MGVideoChannelInSimplex
Video MPEG-4 simplex send	MGVideoChannelOutSimplex
Video H.263 full-duplex	MGH263VideoChannel
Video H.263 simplex receive	MGH263VideoChannelInSimplex
Video H.263 simplex send	MGH263VideoChannelOutSimplex
Video H.264 full-duplex	MGH264VideoChannel
Video H.264 simplex receive	MGH264VideoChannelInSimplex
Video H.264 simplex send	MGH264VideoChannelOutSimplex

Audio channel types

The following table lists the audio channel types most likely to be used in the 3G-324M Interface, and provides their associated identifiers.

Channel type	Identifier
G.723 bypass full-duplex	MGG723BypassChannel
G.723 bypass simplex receive	MGG723BypassChannelInSimplex
G.723 bypass simplex send	MGG723BypassChannelOutSimplex
AMR bypass full-duplex	MGAMRBypassChannel
AMR bypass simplex receive	MGAMRBypassChannelInSimplex
AMR bypass simplex send	MGAMRBypassChannelOutSimplex
AMR channel (for AMR transcoding) full-duplex	MGAMRChannel
AMR channel (for AMR transcoding) simplex encoding (direction to terminal)	MGAMRChannelEncodeSimplex
AMR channel (for AMR transcoding) simplex decoding (direction from terminal)	MGAMRChannelDecodeSimplex
G.711 channel (for AMR transcoding) full-duplex	G711FullDuplex
G.711 channel (for AMR transcoding) simplex receive	G711EncodeSimplex
G.711 channel (for AMR transcoding) simplex send	G711DeodeSimplex

Example

The following example shows how to create a video channel:

```
GwConfig[i].VidChan.Addr.channelType            = MGVideoChannel;
GwConfig[i].VidChan.Param.channelType           = MGVideoChannel;
ret = mspCreateChannel( GwConfig[i].VidChan.ctahd,
&GwConfig[i].VidChan.Addr,
&GwConfig[i].VidChan.Param,
&GwConfig[i].VidChan.hd);
```

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

Connecting endpoints and channels

Connect endpoints and channels by using **mspConnect** as shown in the following example:

```
mspConnect ( EP1 handle, channel handle, EP2 handle)
```

For video-specific endpoints, the order of the endpoints in **mspConnect** is important. The video RTP endpoint is endpoint 1, and the MUX endpoint (MSP_ENDPOINT_MUX) is endpoint 2.

For example:

```
ret = mspConnect(   GwConfig[i].VidRtpEp.hd,
GwConfig[i].VidChan.hd,
GwConfig[i].MuxEp.hd);
```

You can configure three types of connections for video endpoints:

Connection	Description
Simplex	Create a simplex connection by creating a simplex channel and connecting it to an RTP video simplex endpoint at one end and a MUX endpoint at the other end. This connection carries media in a single direction.
Symmetrical duplex	Create a duplex connection by creating a full-duplex channel and connecting it to an RTP video full-duplex endpoint at one end and a MUX endpoint at the other end. This connection carries media in two directions.
RTP multiple unicast	Create a series of RTP simplex send video endpoints (up to eight) and connect all of these RTP endpoints to a MUX endpoint through a video simplex send channel.

Whether using the pass-through configuration or the audio transcoding configuration, there are always two connections to the MUX endpoint: one for the video channel and one for the audio channel. Create these connections by using **mspConnect** in a serial fashion. Wait for MSPEVN_CONNECT_DONE for the first channel before invoking **mspConnect** for the second channel.

The same limitation applies when disconnecting the channels from the MUX endpoint. Wait for MSPEVN_DISCONNECT_DONE for the first channel before calling **mspDisconnect** for the second channel.

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

Setting up tracing and monitoring functionality

Use the following functions to set up tracing and monitoring functionality for the 3G-324M Interface:

Function	Description	Associated Events
h324SetTrace	Defines the level of tracing for the 3G-324M library.	None
h324LineErrorReporting	Turns on error reporting statistics in the H.223 demultiplexer. Use these statistics to determine the quality of the inbound radio link from the H.324 terminal to the demultiplexer.	MSPEVN_DEMUX_CRC_ERR_REPORT MSPEVN_DEMUX_PERIODIC_STATS

Stopping a call or session

Use the following functions to stop a call or session:

Function	Description.
h324CloseChannel	Closes an existing media channel (audio or video). The H.324M Middleware returns one of the following events: H324EVN_CHANNEL_CLOSED H.324M Middleware closed the specified channel. H324EVN_CHANNEL_CLOSE_FAILED The remote terminal rejected the request to close the channel. **Note:** Closing media channels when terminating a call is not required.
h324EndSession	Terminates the current H.324 session at the end of a call, by sending an EndSession message to the remote terminal. This H.324M Middleware returns one of the following events: H324EVN_END_SESSION_DONE H.324M Middleware ended the current H.324 session. H324EVN_END_SESSION_TIMER_EXPIRED The remote terminal did not respond properly to the H.324 end session command. Pass this event to the H.324M Middleware. The Middleware ends the session and then returns an H324_END_SESSION_DONE event.
h324Stop	Stops the H.324M Middleware for the specified MUX endpoint. This function must be called at the end of an H.324 call.
h324Delete	Cleanly releases any objects and memory associated with an instance of the H.324M Middleware.

Negotiating H.263+ profile 3

As documented in 3GPP TR 26.911, optional annexes of ITU-T Recommendation H.263 are useful for improving the compression efficiency and error resilience of the codec. 3GPP recommends that an ITU-T Recommendation H.263 video decoder support the following annexes:

- Annex I (Advanced INTRA Coding mode) improves error resilience and compression efficiency.
- Annex J (Deblocking Filter mode) improves compression efficiency.
- Annex K (Slice Structured mode without Rectangular Slice submode) improves error resilience.
- Annex T (Modified Quantization mode) improves compression efficiency.

These annexes characterize the Version 2 (H.263+) Interactive and Streaming Wireless Profile, designated as profile 3 in the H.263 Annex X Recommendation.

3G-324M Interface support for H.263+ profile 3

The 3G-324M Interface supports H.263+ profile 3. To enable H.263+ profile 3 in a Video Access application, set the bit_mask field in the MH263CAPABILITY structure to h263_profile3_enabled (0x0001). The H.324M Middleware then negotiates with the remote terminal to see if it also supports H.263+ profile 3:

If the remote terminal...	Then the H.324M Middleware
Supports H.263+ profile 3	Uses the H.263+ profile 3 codec.
Does not support H.263_ profile 3	Falls back to H.263 baseline (profile 0).

The application can obtain the outcome of the H.263 negotiation by checking the value of the bit_mask field in the MH263CAPABILITYOPTIONS structure. If h263_profile3_enabled (0x0001) is set, an H.263+ profile 3 decoder is required to decode the video stream sent by the peer. The application can then check the advancedIntraCodingMode, deblockingFilterMode, slicesInOrder_NonRect, and modifiedQuantizationMode fields to find out which H.323 annexes are used by the remote terminal.

For more information, see *MH263CAPABILITYOPTIONS structure* on page 39.

Note: The application must use RFC 2429 RTP packetization to transport H.263+ profile 3 video. RFC 2190 cannot be used for this purpose.

H.263+ profile 3 parameters

The H.263+ profile 3 parameters for the 3G-324M Interface are specified in the MH263CAPABILITY and MH263CAPABILITYOPTIONS structures in *avoptions.h*.

MH263CAPABILITY structure

The MH263CAPABILITY structure defines the H.324M local terminal capabilities for the H.263 codec. Set the bit_mask field to h263_profile3_enabled (0x0001) to enable H.263+ profile 3:

```
typedef struct _MH263CAPABILITY
{
    unsigned short  bit_mask;

....

#define h263_profile3_enabled               0x0001

...
}
```

MH263CAPABILITY is a substructure to the MEDIA_UNION structure, which itself is a substructure to the H324_LCD structure and the H324_TERM_CAPS structure. For more information, see *H324_LCD* on page 140 or *H324_TERM_CAPS* on page 144.

MH263CAPABILITYOPTIONS structure

The MH263CAPABILITYOPTIONS structure enables the application to check which H.263+ profile 3 annex (I, J, K, or T) is used by the remote terminal, if H.263+ profile 3 codec was negotiated:

```
typedef struct MH263CAPABILITYOPTIONS
{
     unsigned int bit_mask;
     H245_BOOL advancedIntraCodingMode;      // Annex I
     H245_BOOL deblockingFilterMode;         // Annex J
     H245_BOOL improvedPBFramesMode;
     H245_BOOL unlimitedMotionVectors;

     H245_BOOL fullPictureFreeze;
     H245_BOOL partialPictureFreezeAndRelease;
     H245_BOOL resizingPartPicFreezeAndRelease;
     H245_BOOL fullPictureSnapshot;
     H245_BOOL partialPictureSnapshot;
     H245_BOOL videoSegmentTagging;
     H245_BOOL progressiveRefinement;
     H245_BOOL dynamicPictureResizingByFour;
     H245_BOOL dynamicPictureResizingSixteenthPel;
     H245_BOOL dynamicWarpingHalfPel;
     H245_BOOL dynamicWarpingSixteenthPel;
     H245_BOOL independentSegmentDecoding;
     H245_BOOL slicesInOrder_NonRect;        // Annex K
     H245_BOOL slicesInOrder_Rect;
     H245_BOOL slicesNoOrder_NonRect;
     H245_BOOL slicesNoOrder_Rect;
     H245_BOOL alternateInterVLCMode;
     H245_BOOL modifiedQuantizationMode;     // Annex T
     H245_BOOL reducedResolutionUpdate;
} MH263CAPABILITYOPTIONS;
```

MH263CAPABILITYOPTIONS is a substructure to the H324_LCD structure, which is carried with the video OLC event (H324EVN_LCD). For more information, see *H324_LCD* on page 140.

4 PSTN and IP network interactions during a call

Sending control messages to the remote terminal

Use these functions to send control messages to the remote terminal:

Function	Description
h324_h223SkewIndication	Sends an H.223 skew indication to the 3G-324M remote terminal to advertise any known skew between the audio and video media streams sent by the H.324 Interface.
h324RoundTripDelay	Sends a round trip delay command to the remote terminal.
h324UserIndication	Sends a user indication to the remote terminal. (The remote terminal can interpret the messages according to a proprietary scheme.)
h324VideoFastUpdate	Sends a video fast update message to the remote terminal to request that the terminal generate a complete intra-coded picture.
h324VideoTemporalSpatialTradeoff	Sends a VideoTemporalSpatialTradeoff indication to the remote terminal.
h324ModifyOutVideoChannelParam	Closes an existing video logical channel and then re-opens the channel using user-supplied logical channel parameters.
h324VendorIDIndication	Sends a VendorID indication to the remote terminal. The application can build a vendor indication identifier.

Transferring DTMF digits according to RFC 2833

You can transfer DTMF digits according to RFC 2833 through audio pass-through channels, as well as through encoder and decoder channels. The procedure for transferring DTMF digits differs by channel type, as described in this topic.

Transferring DTMF digits through audio pass-through channels

This subsection describes how to transfer DTMF digits through audio pass-through channels.

From the PSTN side to the IP side

Use **h324PassthruPlayRFC2833** to transfer a DTMF digit through an audio pass-through channel from the PSTN side to the IP side of the 3G-324M Interface. This function does the following:

- Creates an RFC 2833 DTMF event, which packages the DTMF tone information into a specially formatted RTP packet.
- Transfers the packet to the IP network.

The following example shows how to transfer a DTMF digit in this way:

```
{
    DWORD result;
    WORD nEventID, nEventDuration;
    nEventID = ‘3’; // The keypad digit ‘3’
    EventDuration = 80; // Digit pressed for 80 ms.
    result = h324PassthruPlayRFC2833(GwConfig[0].MuxEp.hd, nEventID, EventDuration);
    if (result != SUCCESS)
        printf("ERROR: h324PassthruPlayRFC2833() failed.\n");
}
```

From the IP side to the PSTN side

When a DTMF digit arrives from the IP side of the 3G-324M Interface in RFC 2833 encapsulation, the receiving RTP endpoint conveys it to the application in an MSPEVN_RFC2833_REPORT event. It is the responsibility of the application to forward that digit to the 3G-324M terminal through an H.245 user indication message (**h324UserIndication**), if required.

The DTMF represented by the MSPEVN_RFC2833_REPORT is not forwarded automatically to the 3G-324M terminal in the audio channel because that channel carries compressed audio (AMR or G.723.1). This is unreliable for transmission of DTMF tones.

For more information about the MSPEVN_RFC2388_REPORT event, see the *Fusion Developer's Manual*.

Transferring DTMF digits through audio encoder and decoder channels

The ability to transfer DTMF digits according to RFC 2833 through audio encoder and decoder channels is provided by Fusion MSPP functionality. For information about RFC 2833 support for these channel types, see the *Fusion Developer's Manual*.

5 Querying and modifying the configuration

MSPP queries and commands for the 3G-324M Interface

This topic describes the following queries and commands to use with the 3G-324M Interface:

- Queries for all video endpoints
- Query for H.264 endpoints
- Query for the video channel jitter buffer filters
- Commands for all video endpoints
- Commands for MPEG-4 and H.264 endpoints only
- Commands for H.263 endpoints only
- Commands for the video channel jitter buffer filters

Queries for all video endpoints

The following MSPP queries can be sent to MPEG-4, H.263, and H.264 RTP endpoints:

Query (associated structure)	**Description**
MSP_QRY_RTPFDX_STATUS MSP_QRY_RTPIN_STATUS MSP_QRY_RTPOUT_STATUS (msp_ENDPOINT_RTPFDX_STATUS)	Returns information about the specified full-duplex or simplex RTP endpoint filter state. For more information, see the *MSPP Developer's Reference Manual*.
MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL MSP_QRY_RTPOUT_VIDEO_RTP_PKTSZ_CTRL (msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL)	Returns the packet size and aggregation parameter settings for the packets transmitted by the specified full-duplex or simplex send RTP endpoint from the PSTN to the IP side of the gateway. For more information, see *Adjusting RTP packetization parameters* on page 52.

Query for H.264 endpoints

The following MSPP query can be sent to H.264 RTP endpoints:

Query (associated structure)	**Description**
MSP_QRY_RTPFDX_H264_TX_STATUS MSP_QRY_RTPOUT_H264_TX_STATUS (msp_ENDPOINT_RTPFDX_H264_TX_STATUS)	Returns transmit packetization status information for H.264 full duplex and simplex send endpoints. For more information, see *Querying an H.264 endpoint for transmit status* on page 67.

Query for the video channel jitter buffer filters

The following MSPP query can be sent to the video channel jitter buffer filters:

Query (associated structure)	**Description**
MSP_QRY_JITTER_VIDEO_GET_STATE (msp_FILTER_JITTER_VIDEO_STATE)	Returns the status of the video jitter buffer. For more information, see *Querying the video jitter buffer state* on page 63.

Commands for all video endpoints

The following MSPP commands can be sent to MPEG-4, H.263, and H.264 video RTP endpoints:

Command (associated structure)	**Description**
MSP_CMD_RTPFDX_CALC_SKEW_OFFSET MSP_CMD_RTPIN_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 RTP endpoints. For more information, see *Using RTCP for audio/video synchronization* on page 58.
MSP_CMD_RTPFDX_CONFIG MSP_CMD_RTPIN_CONFIG MSP_CMD_RTPOUT_CONFIG (msp_ENDPOINT_RTPFDX_CONFIG)	Sets configuration parameters for the specified full-duplex or simplex RTP endpoint. This command also sets timestamp frequencies for RTP endpoints. For more information, see the *MSPP Developer's Reference Manual* and *Setting timestamp frequencies for full-duplex RTP endpoints* on page 54.
MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL MSP_CMD_RTPIN_IFRAME_NOTIFY_CTRL MSP_CMD_RTPOUT_IFRAME_NOTIFY_CTRL (msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL)	Enables or disables I-frame notification for the specified full-duplex or simplex RTP endpoint. For more information, see *Enabling I-frame notification* on page 49.
MSP_CMD_RTPFDX_LINK_EVENTS MSP_CMD_RTPOUT_LINK_EVENTS (msp_ENDPOINT_RTPFDX_LINK_EVENTS)	Enables or disables reporting link availability transitions through unsolicited events for the specified full-duplex or simplex send RTP endpoint. For more information, see the *MSPP Developer's Reference Manual*.
MSP_CMD_RTPFDX_MAP MSP_CMD_RTPIN_MAP (msp_ENDPOINT_RTPFDX_MAP)	Assigns a payload ID to a vocoder for the specified full-duplex or simplex receive RTP endpoint. For more information, see the *MSPP Developer's Reference Manual*.

Command **(associated structure)**	**Description**
MSP_CMD_RTPFDX_RTCP_EVENTS MSP_CMD_RTPIN_RTCP_EVENTS MSP_CMD_RTPOUT_RTCP_EVENTS (msp_ENDPOINT_RTPFDX_RTCP_EVENTS)	Enables or disables reporting RTCP events through unsolicited events for the specified full-duplex or simplex RTP endpoint. For more information, see the *MSPP Developer's Reference Manual*.
MSP_CMD_RTPFDX_SET_VID_TX_PID MSP_CMD_RTPOUT_ SET_VID_TX_PID (msp_ENDPOINT_RTPFDX_SET_VID_TX_PID	Sets a transmit payload ID for the specified full-duplex or simplex send video RTP endpoint. For more information, see *Setting the video transmit RTP payload ID* on page 57.
MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL MSP_CMD_RTPOUT_VIDEO_RTP_PKTSZ_CTRL (msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL)	Sets the maximum packet size and aggregation threshold, and indicates whether aggregation takes place for the packets transmitted by the specified full-duplex or simplex send RTP endpoint from the PSTN to the IP side of the gateway. For more information, see *Adjusting RTP packetization parameters* on page 52.
MSP_CMD_RTPFDX_VIDEO_SKEW_TIME MSP_CMD_RTPOUT_ 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 or simplex send video RTP endpoint. For more information, see *Enabling RTP video send endpoints to send video skew values to the IP destination* on page 61.
MSP_CMD_RTPFDX_RTTS_CTRLMSP_CMD_RTPOUT_ RTTS_CTRL(msp_ENDPOINT_RTPFDX_RTTS_CTRL)	Enables or disables reporting the real-time timestamping feature for the specified full-duplex or simplex send RTP endpoint. For more information, see *Configuring real-time timestamp generation* on page 68.

Command for MPEG-4 and H.264 endpoints only

The following MSPP command can be sent to MPEG-4 and H.264 video RTP endpoints only:

Command **(associated structure)**	**Description**
MSP_CMD_RTPFDX_OUT_OF_BAND_DCI MSP_CMD_RTPOUT_OUT_OF_BAND_DCI (msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI)	Optionally replaces in-band DCI with commanded out-of-band DCI for the specified full-duplex or simplex send RTP endpoint. For more information, see *Inserting out-of-band DCI into the video bit stream* on page 56.

Commands for H.263 endpoints only

The following MSPP commands can be sent to H.263 video RTP endpoints only:

Command (associated structure)	**Description**
MSP_CMD_RTPFDX_DISCARD_PENDING_PFRAMES MSP_CMD_RTPIN_DISCARD_PENDING_PFRAMES	Discontinues forwarding partial frames from IP to PSTN, until an incoming I-frame is detected on the specified full-duplex or simplex receive H.263 RTP endpoint. For more information, see *Discarding leading partial frame video packets* on page 48.
MSP_CMD_RTPFDX_H263_ENCAP_CTRL MSP_CMD_RTPOUT_H263_ENCAP_CTRL (msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL)	Sets the RFC encapsulation type used for the specified full-duplex or simplex send H.263 RTP endpoint. For more information, see *Setting the H.263 RFC encapsulation type* on page 55.
MSP_CMD_RTPFDX_STOP_VIDEO_RX MSP_CMD_RTPIN_STOP_VIDEO_RX	Stops the video transmission in the direction of the PSTN for the specified full-duplex or simplex receive H.263 RTP endpoint, after the current frame has been fully received. For more information, see *Ensuring a smooth transition when the video source changes* on page 51.

Commands for the video jitter buffer filters

The following MSPP commands can be sent to MPEG-4, H.263, and H.264 video channel jitter filters only:

Command (associated structure)	**Description**
MSP_CMD_JITTER_CHG_VIDEO_LATENCY (msp_FILTER_JITTER_VID_LATENCY)	Changes the configured video jitter buffer latency. The maximum latency is 2000 ms (2 seconds). For more information, see *Setting the video jitter buffer latency* on page 62.
MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF	Removes the excess video packets that accumulate above the configured latency in the video jitter buffer. The configured latency value remains in the video jitter buffer after you remove the excess video packets. For more information, see *Removing excess video packets from the video jitter buffer* on page 64.
MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF	Removes all data from the video jitter buffer, including all video packets and the configured video latency value. For more information, see *Purging the video jitter buffer* on page 65.

Creating and sending MSPP queries

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

The mspBuildQuery macro and **mspSendQuery** are described below. The MSPP endpoint queries are described in *MSPP queries and commands for the 3G-324M Interface* on page 43.

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*.

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 from the mspBuildQuery macro.

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

Creating and sending MSPP commands

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

The mspBuildCommand macro and **mspSendCommand** are described below. The MSPP endpoint commands are described in *MSPP queries and commands for the 3G-324M Interface* on page 43.

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*.

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 from 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*.

Modifying the RFC 2833 DTMF configuration

Modify the DTMF mode and payload ID in the RFC 2833 DTMF configuration for pass-through channels, and for encoder and decoder channels:

- Modify the DTMF mode and payload ID for a pass-through channel by using **h324PassthruDTMFMode**.
- Modify the DTMF mode for an encoder or decoder channel by using standard Fusion MSPP Functionality. For more information, see the *Fusion Developer's Manual*.

Discarding leading partial frame video packets

You can request that an endpoint discard incoming leading partial frame video packets to reduce video noise. When you do this, the endpoint discontinues forwarding partial frames in the direction of the PSTN until it detects an I-frame. At that time, the endpoint resumes forwarding the detected I-frame and all subsequent frames.

Use this feature to improve quality where a conversation might be interrupted and then restarted, such as in a video tromboning application.

Note: This feature is for full-duplex and simplex receive H.263 RTP endpoints only. The procedure and example are shown for full-duplex endpoints.

The following table shows how to discard leading partial frame video packets:

Step	Action
1	Disable the RTP endpoint by invoking **mspDisableEndpoint**.
2	Create the MSP_CMD_RTPFDX_DISCARD_PENDING_FRAMES command. This command has no associated structure. For more information see *Creating and sending MSPP commands* on page 47.
3	Enable the RTP endpoint by invoking **mspEnable**.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_DISCARD_PENDING_FRAMES command was successfully sent to the specified RTP endpoint on the CG board.

Example

The following example shows how to discard leading partial frame video packets for the endpoint with the handle ephd:

```
ret = mspDisableEndpoint(ephd);
ret = WaitForSpecificEvent(gw, ctaQueueHd, MSPEVN_DISABLE_ENDPOINT_DONE, &event );
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                     MSP_CMD_RTPFDX_DISCARD_PENDING_PFRAMES), NULL, 0);
ret = WaitForSpecificEvent(gw, ctaQueueHd, (MSPEVN_SENDCOMMAND_DONE
                           SP_CMD_RTPFDX_DISCARD_PENDING_PFRAMES),&event );
ret = mspEnableEndpoint(ephd);
ret = WaitForSpecificEvent(gw, ctaQueueHd, MSPEVN_ENABLE_ENDPOINT_DONE, &event );
```

Enabling I-frame notification

You can configure any type of video RTP endpoint to notify the application when an I-frame is detected, either received from IP and being sent to 324M, or received from 324M and being sent to IP.

For MPEG-4 and H.264 endpoints, if the I-frame reported was preceded by decoder configuration information (DCI), then this DCI is also reported in the returned event. An application can monitor DCI to determine changes in DCI. There may be cases where DCI changes require new signaling (for example, to close or reopen a new video logical channel for 324M upon a change in DCI).

Note: This feature is for full-duplex and simplex video endpoints. The procedure and example are shown for full-duplex endpoints.

To enable I-frame notification, create the MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL command, and send it to the CG board.

The MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL command uses the msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL structure to determine whether to enable or disable the I-frame notification functionality, and to determine when to generate the notifications.

For more information see *Creating and sending MSPP commands* on page 47 and *msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL* on page 89.

The following events can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL command was successfully sent to the specified endpoint on the CG board.
MSPEVN_VIDEO_IFRAME_RTPIP	An I-frame has come from the IP port.
MSPEVN_VIDEO_IFRAME_H324	An I-frame has come from the PSTN port.

If DCI is being reported with one of these I-frame events, the returned event buffer contains the following structure:

```
typedef struct
{
DWORD   FilterId;
DWORD   dciLen;                     //Length of DCI buffer
U8      dciData[MAX_DCI_BUF_SIZE];  //DCI data buffer (max 255 bytes)
} msp_RTP_VIDEO_DCI;
```

Example

The following example shows how to enable I-frame notification in the 324M-to-IP direction for the H.264 full-duplex endpoint using the MSP handle ephd and to retrieve the returned DCI in the event buffer:

```
msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL notifyCmd;
notifyCmd.h324Notify = H2NMS_DWORD(1);
notifyCmd.ipNotify = H2NMS_DWORD(0);
command = mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H264,
                          MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL);
ret = mspSendCommand(ephd, command, &notifyCmd, sizeof(notifyCmd));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
                 MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
. . .
//Process returned I-Frame unsolicited event
case MSPEVN_VIDEO_IFRAME_H324:
  if(GwConfig[nGw].vidRtpEp[EP_OUTB].hd == pevent->objHd ||
  GwConfig[nGw].vidRtpEp[EP_INB].hd == pevent->objHd)
  {
      printf("Unsol Event MSPEVN_VIDEO_IFRAME_H324 %s\n", event_val);
      //Unsol events always return a buffer of at least 4 bytes for Filter ID field
      if (pevent->buffer != NULL && pevent->size > 4)
      {
          U32 dciLength;
          msp_RTP_VIDEO_DCI *pDCIEvent;
          pDCIEvent = (msp_RTP_VIDEO_DCI*)(pevent->buffer);
          dciLength = (U32)NMS2H_DWORD(pDCIEvent->dciLen);
          if (dciLength > 0)
          {
              U8 *pDCIdata = pDCIEvent->dciData;
              printf("DCI DATA (size = %d): \n", dciLength);
              for (U32 i=0; i<dciLength; i++)
              {
                  if(i%16 == 0)
                      printf("%02x: ",i);
                  printf(" %02x", pDCIdata[i]);
                  if (i%16 == 15)
                      printf("\n");
              }
              printf("\n");
          }
      }
  }
  break;
```

Ensuring a smooth transition when the video source changes

Ensure a smooth transition when the video source changes by stopping the video transmission in the direction of the PSTN after the current frame has been fully received. This ensures that the 3G-324 Interface sends the video data to the remote terminal at a frame boundary.

After you receive confirmation that the video transmission has been stopped (through the MSPEVN_VIDEO_RX_STOPPED event), you can safely disable the RTP endpoint, if necessary.

Note: This feature is for full-duplex and simplex receive H.263 RTP endpoints only. The procedure and example are shown for full-duplex endpoints.

To ensure a smooth transition when the video source changes, create the MSP_CMD_RTPFDX_STOP_VIDEO_RX command, and send it to the CG board. This command has no associated structures.

For more information see *Creating and sending MSPP commands* on page 47.

The following events can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	The MSP_CMD_RTPFDX_STOP_VIDEO_RX command was successfully sent to the specified RTP endpoint on the CG board.
MSPEVN_VIDEO_RX_STOPPED	Indicates one of the following: • A frame boundary has been detected in the video flowing through the RTP endpoint from the IP side to the MUX, after the application issued the MSP_CMD_RTPFDX_STOP_VIDEO_RX command. • The video RTP endpoint timed out awaiting a frame boundary after the application issued the MSP_CMD_RTPFDX_STOP_VIDEO_RX command.

Example

The following example shows how to stop video transmission for the endpoint with the MSP handle ephd:

```
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                     MSP_CMD_RTPFDX_STOP_VIDEO_RX), NULL, 0);
ret = WaitForSpecificEvent(gw, ctaQueueHd,
      (MSPEVN_SENDCOMMAND_DONE |
      MSP_CMD_RTPFDX_STOP_VIDEO_RX),
      &event );
ret = WaitForSpecificEvent(gw, ctaQueueHd,
      MSPEVN_VIDEO_RX_STOPPED,
      &event );
```

Adjusting RTP packetization parameters

Adjust the RTP packetization parameters for data transmitted from the PSTN side to the IP side of the 3G-324M Interface. You can do this by setting parameters to control the maximum size and the aggregation threshold of the payload in the RTP packets created by the video RTP endpoint. You can also set a parameter that controls whether packet aggregation takes place - except on H.264 endpoints where there is no aggregation.

The size and aggregation parameters are used to fine tune CG board performance against system delay, network traffic, and so forth. An example is with video tromboning, where the system delay is doubled because of the application architecture. In this case, the goal is to reduce the RTP packet size, while maintaining minimum density requirements.

Note: NMS recommends using default parameter values for most applications.

You can optionally query existing parameter settings, before or after you reset parameter values. If you perform the query as part of a read-modify-write sequence, call the query after you disable the endpoint that you want to modify.

Note: This feature is for all video full-duplex and simplex send RTP endpoints only. The procedure and example are shown for full-duplex endpoints.

The following table shows how to query and change RTP packet size and aggregation values:

Step	Action
1	Create the MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL query, and send it to the CG board. The MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL query uses the msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL structure to display parameters for the specified RTP endpoint. For more information, see *Creating and sending MSPP queries* on page 47 and *msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL* on page 91.
2	Disable the RTP endpoint by invoking **mspDisableEndpoint**.
3	Create the MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL command, and send it to the CG board. The MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL command uses the msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL structure to change parameter values for the specified RTP endpoint.
4	Enable the RTP endpoint by invoking **mspEnableEndpoint**.

The following events can be returned:

Event	Description
MSPEVN_QUERY_DONE	Indicates that the MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL query was successfully sent to the specified RTP endpoint on the CG board.
MSPEVN_SENDCOMMAND_DONE	Indicates that the MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL command was successfully sent to the specified RTP endpoint on the CG board.

Examples

The following example shows how to query the values of the packet size and aggregation parameters for the packets transmitted by the specified RTP endpoint with the MSP handle ephd:

```
msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL pktSzCtrl;
DWORD query
query = mspBuildQuery(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                      MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL);
if (mspSendQuery(ephd, query) != SUCCESS)
    printf("\n\t ERROR: mspSendQuery failed.\n");
expectedEvent = (MSPEVN_QUERY_DONE | MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendQuery failed to send valid completion event\n");
pktSzCtrl = *(msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL *)(((char
*)event.buffer)+sizeof(DWORD));
pktSzCtrl.pktMaxSz = NMS2H_DWORD(pktSzCtrl.pktMaxSz);
pktSzCtrl.aggThreshold = NMS2H_DWORD(pktSzCtrl.aggThreshold);
pktSzCtrl.enableAggregation = NMS2H_DWORD(pktSzCtrl.enableAggregation);
printf("\n\tQuery RTP Maximum Packet Size Control values:\n");
printf("\t\tRTP maximum packet size:   %d bytes\n", pktSzCtrl.pktMaxSz);
printf("\t\tRTP aggregation flag = %d\n", pktSzCtrl.enableAggregation);
if (pktSzCtrl.enableAggregation)
    printf("\t\tRTP aggregation threshold: %d bytes\n", pktSzCtrl.aggThreshold);
// Query returns a CTA buffer; release it.
mspReleaseBuffer(ctaQueueHd, event.buffer);
```

The following example shows how to adjust the maximum size and the threshold for an RTP endpoint with the MSP handle ephd:

```
CTA_EVENT   event;
msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL pktSzCtrl;
pktSzCtrl.pktMaxSz          = 1400;  // Random example: max – 40 bytes
pktSzCtrl.aggThreshold      = 600;   // Random example: default – 100 bytes
pktSzCtrl.enableAggregation = 1;     // ENABLE aggregation of GOB frames
ret = mspDisableEndpoint(ephd);
ret = WaitForSpecificEvent(gw, ctaQueueHd, MSPEVN_DISABLE_ENDPOINT_DONE, &event );
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                     MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL), &pktSzCtrl,
                     sizeof(pktSzCtrl) );
ret = WaitForSpecificEvent(gw, ctaQueueHd,
MSPEVN_SENDCOMMAND_DONE | MSP_CMD_RTPFDX_VIDEO_RTP_PKTSZ_CTRL,
                          &event );
ret = mspEnableEndpoint(ephd);
ret = WaitForSpecificEvent(gw, ctaQueueHd, MSPEVN_ENABLE_ENDPOINT_DONE, &event );
```

Setting timestamp frequencies for full-duplex RTP endpoints

Set the timestamp frequency for an RTP endpoint by changing the rtpTsFreq parameter in the MSP_CMD_RTPFDX_CONFIG command.

The default timestamp frequency is 90,000 Hz. To set a frequency value other than the default, use a value other than zero or all ones (0xFFFFFFFF).

Note: This feature is for all video full-duplex and simplex send RTP endpoints. The procedure and example are shown for full-duplex endpoints.

The following table describes how to set timestamp frequencies:

Step	Action
1	Disable the RTP endpoint by invoking **mspDisableEndpoint**.
2	Create the MSP_CMD_RTPFDX_CONFIG command, and send it to the CG board. The MSP_CMD_RTPFDX_CONFIG command uses the standard MSPP msp_ENDPOINT_RTPFDX_CONFIG structure to set the timestamp frequency. For more information, see *Creating and sending MSPP commands* on page 47 and the *MSPP Service Developer's Reference Manual*.
3	Enable the RTP endpoint by invoking **mspEnable**.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_CONFIG command was successfully sent to the specified RTP endpoint on the CG board.

Example

The following example shows how to set the timestamp frequency for an RTP endpoint with the MSP handle ephd:

```
msp_ENDPOINT_RTPFDX_CONFIG mConfig;
CTA_EVENT event;
DWORD ret, expectedEvent
memset(&mConfig, 0xFF, sizeof(mConfig));
mConfig.rtpTsFreq = H2NMS_DWORD(97000) ;

ret = mspDisableEndpoint(ephd);
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                     MSP_CMD_RTPFDX_CONFIG), &mConfig, sizeof(mConfig));
if (ret != SUCCESS)
    printf("\n\t ERROR: mspSendCommand returned failure.\n");
expectedEvent = (MSPEVN_SENDCOMMAND_DONE | MSP_CMD_RTPFDX_CONFIG);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
ret = mspEnableEndpoint(ephd);
```

Setting the H.263 RFC encapsulation type

Configure an H.263 endpoint to use either RFC 2190 or RFC 2429 encapsulation when transmitting H.263 RTP packets. The default encapsulation type is RFC 2190.

H.263 endpoints can accept both RFC 2190 and RFC 2429 packets when receiving data from IP, regardless of how the endpoints are configured. H.263 decapsulation is performed based on the incoming payload ID, and not on the encapsulation type the endpoint is configured to transmit.

Note: This feature is for all video full-duplex and simplex send H.263 RTP endpoints. The procedure and example are shown for full-duplex endpoints.

The following table describes how to set the RFC encapsulation type used by an H.263 RTP endpoint:

Step	Action
1	Disable the RTP endpoint by invoking **mspDisableEndpoint**.
2	Create the MSP_CMD_RTPFDX_H263_ENCAP_CTRL command, and send it to the CG board. The MSP_CMD_RTPFDX_H263_ENCAP_CTRL command uses the msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL structure to set the encapsulation type. For more information, see *Creating and sending MSPP commands* on page 47 and *msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL* on page 88.
3	Enable the RTP endpoint by invoking **mspEnable**.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_H263_ENCAP_CTRL command was successfully sent to the specified RTP endpoint on the CG board.

Example

The following example shows how to change the H.263 encapsulation type for an RTP endpoint with the MSP handle ephd from the default of RFC 2190 to RFC 2429 encapsulation:

```
msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL cmd;
CTA_EVENT event;
DWORD ret, expectedEvent;
cmd.h263Encap = H2NMS_DWORD(MSP_H263_2429) ;
ret = mspDisableEndpoint(ephd);
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
                     MSP_CMD_RTPFDX_H263_ENCAP_CTRL), &cmd, sizeof(cmd));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE | MSP_CMD_RTPFDX_H263_ENCAP_CTRL);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
ret = mspEnableEndpoint(ephd);
```

Inserting out-of-band DCI into the video bit stream

Configure an MPEG-4 or H.264 RTP endpoint to optionally replace in-band DCI (Decoder Configuration Information) with out-of-band DCI, either the first time that in-band DCI is detected in the bit stream, or every time that in-band DCI is detected in the bit stream. With an MPEG-4 endpoint, you can also insert the out-of-band DCI before each transmitted I-frame.

In-band DCI may or may not be part of an MPEG-4 or H.264 bit stream received from a 3G-324M terminal. Even if there is DCI in-band, it may not be reliable, because it is transmitted over a connectionless and possibly error-prone wireless channel. It may be necessary for a video decoder behind the 3G-324M Interface to receive in-band DCI, especially if there is no method for that decoder to receive DCI out-of-band.

Note: This feature is for full-duplex and simplex send MPEG-4 and H.264 RTP endpoints only. The procedure and example are shown for full-duplex endpoints.

An MPEG-4 or H.264 RTP endpoint can be configured to insert DCI received by the application out-of-band into the RTP-packetized bit stream. The insertion will be based on the mode field in the msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI structure.

The following table shows how to insert out-of-band DCI into the MPEG-4 bit stream:

Step	Action
1	Disable the RTP endpoint by invoking **mspDisableEndpoint**.
2	Create the MSP_CMD_RTPFDX_OUT_OF_BAND_DCI command, and send it to the CG board. The MSP_CMD_RTPFDX_OUT_OF_BAND_DCI command uses the msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI structure to send DCI information. For more information, see *Creating and sending MSPP commands* on page 47 and *msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI* on page 90.
3	Enable the RTP endpoint by invoking **mspEnable**.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_OUT_OF_BAND_DCI command was successfully sent to the specified endpoint on the CG board.

Example

The following example shows how to configure an MPEG-4 endpoint with out-of-band DCI for an RTP endpoint with the MSP handle ephd:

```
msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI  notifyCmd;
ret = mspDisableEndpoint(ephd);
command = mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_MPEG4,
                          MSP_CMD_RTPFDX_OUT_OF_BAND_DCI);
for(i = 0;i<= GwConfig[0].DCI.len;i++)
    notifyCmd.data[i] = GwConfig[0].DCI.data[i];//copy DCI from gw
notifyCmd.len =H2NMS_DWORD(GwConfig[0].DCI.len);
notifyCmd.mode = 1;

ret = mspSendCommand(ephd, command, &notifyCmd, sizeof(notifyCmd));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE | MSP_CMD_RTPFDX_OUT_OF_BAND_DCI);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &event) != 0)
    printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
ret = mspEnableEndpoint(ephd);
```

Setting the video transmit RTP payload ID

Set the transmit payload ID for a video RTP endpoint to something other than the default value. The default payload ID values are:

- 100 (MPEG-4)
- 34 (H.263 using RFC 2190)
- 101 (H.263 using RFC 2429)
- 104 (H.264)

To set the transmit payload ID, create the MSP_CMD_RTPFDX_SET_VID_TX_PID command, and send it to the CG board.

The MSP_CMD_RTPFDX_ SET_VID_TX_PID command uses the msp_ENDPOINT_RTPFDX_ SET_VID_TX_PID structure to set the payload ID.

Note: This feature is for all video full-duplex and simplex send RTP endpoints. The procedure and example are shown for full-duplex endpoints.

For more information, see *Creating and sending MSPP commands* on page 47 and *msp_ENDPOINT_RTPFDX_SET_VID_TX_PID* on page 93.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_SET_VID_TX_PID command was successfully sent to the specified RTP endpoint on the CG board.

Example

The following example shows how to set the transmit payload ID for an RTP endpoint with the MSP handle ephd:

```
msp_ENDPOINT_RTPFDX_SET_VID_TX_PID pidCmd;
pidCmd.txPayloadId= H2NMS_DWORD(111) ;
ret = mspSendCommand(ephd, mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
       MSP_CMD_RTPFDX_SET_VID_TX_PID), &pidCmd, sizeof(pidCmd));
if (ret != SUCCESS)
printf("\n\t ERROR: mspSendCommand returned failure.\n");
expectedEvent = (MSPEVN_SENDCOMMAND_DONE | MSP_CMD_RTPFDX_SET_VID_TX_PID);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &event) != 0)
printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

Using RTCP for audio/video synchronization

Use RTCP to communicate skew information to the host application and IP destination through RTP endpoints. To use RTCP, 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 examples are shown for full-duplex endpoints.

During endpoint creation

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

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.

Configure both endpoints in an audio/video stream pair to obtain meaningful skew data. For more information, see *Calculating audio/video skew* on page 59.

For more information about creating and configuring RTP endpoints, see *RTPRTCP_ENDPOINT_PARMS* on page 97 or *RTPRTCP_V6_ENDPOINT_PARMS* on page 99, the *MSPP Developer's Reference Manual*, and the *Fusion Developer's Manual*.

After endpoint creation

After creating endpoints, send the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command to an endpoint so that it can detect skew offset values for incoming data streams and report them 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.

Send the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command to both endpoints in an audio/video stream pair to obtain meaningful audio/video skew data. For more information, see *Calculating audio/video skew* on page 59.

The MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command can return the following events:

Event	Description
MSPEVN_SENDCOMMAND_DONE	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 more information, see *Creating and sending MSPP commands* on page 47.

Calculating audio/video skew

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

Stage	Process
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 behind the audio. A negative result means the audio lags behind the video.
6	Application does either of the following: • Corrects 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*. For information about setting the video jitter buffer latency, see *Setting the video jitter buffer latency* on page 62. • Signals the audio/video skew value to a receiving 3G-324M terminal by using **h324_h223SkewIndication**.

Examples

The following example shows how to use the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command so that the video endpoint with the MSP handle ephd detects 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_H263,
                          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 detects 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 after receiving 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 video send endpoints to send video skew values to the IP destination

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

During endpoint creation

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

Use the following macros to configure the startRtcp bit field:

Use this macro...	To...
RTCP_ENABLE	Enable RTCP for the endpoint.
RTCP_SET_0_INTERVAL	Determine how quickly the RCTP Sender Reports begin after the RTP stream begins.
RTCP_VIDEO_LEADS_AUDIO	Indicate to the application that video leads audio (if true).
RTCP_VIDEO_SKEW	Set the actual video skew time, in ms.

For more information about creating and configuring RTP endpoints, see *RTPRTCP_ENDPOINT_PARMS* on page 97 or *RTPRTCP_V6_ENDPOINT_PARMS* on page 99, the *MSPP Developer's Reference Manual*, and the *Fusion Developer's Manual*.

After endpoint creation

After creating endpoints, send the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command to a video endpoint so that the endpoint sends video skew values to the IP destination 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	MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command was successfully sent to the specified endpoint.

For more information, see *Creating and sending MSPP commands* on page 47.

Example

The following example shows how to send the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command to the video endpoint with the MSP handle ephd so that the endpoint signals 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_H263,
                          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");
```

Managing the video jitter buffer

You can do all of the following to manage the video jitter buffer:

- Set the jitter buffer latency
- Query the jitter buffer state
- Remove excess video packets from the jitter buffer
- Purge the jitter buffer
- Respond to an MSPEVN_REACH_VIDEOLATENCY_LIMIT event

Setting the video jitter buffer latency

Note: This feature is for MPEG-4, H.263, and H.264 channels. The example shown is for H.263.

You can set the video jitter buffer latency to any value less than 2000 ms (2 seconds). To set the video jitter buffer latency, send the MSP_CMD_JITTER_CHG_VIDEO_LATENCY the CG board.

The MSP_CMD_JITTER_CHG_VIDEO_LATENCY command uses the msp_FILTER_JITTER_VID_LATENCY structure to set the jitter buffer latency value.

For more information, see *Creating and sending MSPP commands* on page 47. and *msp_FILTER_JITTER_VID_LATENCY* on page 93.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_JITTER_CHG_VIDEO_LATENCY command was successfully sent to the specified jitter buffer filter on the CG board.

Example

The following example shows how to change the jitter video latency:

```
msp_FILTER_JITTER_VID_LATENCY  jitter_vid_latncy;
MSPHD hObject;
jitter_vid_latncy.value = H2NMS_DWORD(vid_latency); //vid_latency is the value in mSec
if((ret = mspSendCommand( hObject, mspBuildCommand(MSP_FILTER_JITTER_VIDEO_H263,
    MSP_CMD_JITTER_CHG_VIDEO_LATENCY),(void*)&jitter_vid_latncy,
sizeof(jitter_vid_latncy) ))
    != SUCCESS )
{
printf("mspSendCommand() failed with  ret=%x", ret);
return FAILURE;
}
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
     MSP_CMD_JITTER_CHG_VIDEO_LATENCY);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

Querying the video jitter buffer state

Note: This feature is for MPEG-4, H.263, and H.264 channels. The example shown is for H.263.

You can query the video jitter buffer filter, before or after you change the latency value.

To query the video jitter buffer, create the MSP_QRY_JITTER_VIDEO_GET_STATE query, and send it to the CG board. The MSP_QRY_JITTER_VIDEO_GET_STATE query uses the msp_FILTER_JITTER_VIDEO_STATE structure to display parameters for the jitter buffer.

For more information, see *Creating and sending MSPP queries* on page 47 and *msp_FILTER_JITTER_VIDEO_STATE* on page 94.

The following events can be returned:

Event	Description
MSPEVN_QUERY_DONE	MSP_QRY_JITTER_VIDEO_GET_STATE query was successfully sent to the specified jitter buffer filter on the CG board.

Example

The following example shows how to query the video jitter filter:

```
DWORD l_nVideoFilterType;
DWORD query
l_nVideoFilterType = MSP_FILTER_JITTER_VIDEO_H263;
query = mspBuildQuery(l_nVideoFilterType, MSP_QRY_JITTER_VIDEO_GET_STATE);
if (mspSendQuery(ephd, query) != SUCCESS)
    printf("\n\t ERROR: mspSendQuery failed.\n");
expectedEvent = (MSPEVN_QUERY_DONE | MSP_QRY_JITTER_VIDEO_GET_STATE);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &pevent) != 0)
    printf("\n\tERROR: mspSendQuery failed to send valid completion event\n");
printf("\n pktsReceived = %8d, pktsAccepted = %8d, pktsRejected = %8d,
lates  = %8d, pktsLost = %8d, pktsCurrent  = %8d,  overflows    = %8d,
underflows   = %8d, duplicates = %8d, reorders  = %8d, jitterBufDelay  = %8d milliSec,
configuredLatency = %8d milliSec, h264AggPkts  = %8d, h264FragPkts = %8d, pktsDiscarded =
%8d",
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->pktsReceived  ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->pktsAccepted  ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->pktsRejected  ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->lates         ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->pktsLost      ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->pktsCurrent   ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->overflows     ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->underflows    ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->duplicates    ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->reorders      ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->jitterBufDelay),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))->configuredLatency),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))-> h264AggPkts  ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))-> h264FragPkts ),
NMS2H_DWORD(((msp_FILTER_JITTER_VIDEO_STATE*)(pevent->buffer))-> pktsDiscarded ));
mspReleaseBuffer(ctaQueueHd, pevent.buffer);
```

Removing excess video packets from the video jitter buffer

Note: This feature is for MPEG-4, H.263, and H.264 channels. The example shown is for H.263.

You can remove the excess video packets that accumulate above the configured latency in the video jitter buffer. The data for the configured latency duration remains in the video jitter buffer after you remove the excess video packets.

Procedure

To remove excess video packets from the video jitter buffer, create the MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF command, and send it to the CG board.

For more information, see *Creating and sending MSPP commands* on page 47.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF command was successfully sent to the specified jitter buffer filter on the CG board.

Example

The following example shows how to remove excess video packets from the video jitter buffer:

```
MSPHD hObject;
if((ret = mspSendCommand( hObject, mspBuildCommand(MSP_FILTER_JITTER_VIDEO_H263,
MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF ),
(void*)0, 0 )) != SUCCESS )
{
printf("mspSendCommand() failed with  ret=%x", ret);
return FAILURE;
}
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
     MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

Purging the video jitter buffer

Note: This feature is for MPEG-4, H.263, and H.264 channels. The example shown is for H.263.

Purging the video jitter buffer removes all data regardless of the configured video jitter buffer latency value.

To purge the video jitter buffer, create the MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF command, and send it to the CG board.

For more information, see *Creating and sending MSPP commands* on page 47.

The following events can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF command was successfully sent to the specified RTP endpoint on the CG board.

Example

The following example shows how to purge the video jitter buffer:

```
MSPHD hObject
if((ret = mspSendCommand( hObject, mspBuildCommand(MSP_FILTER_JITTER_VIDEO_H263,
MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF ),
(void*)0, 0 )) != SUCCESS )
{
printf("mspSendCommand() failed with  ret=%x", ret);
return FAILURE;
}
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
     MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

Responding to an MSPEVN_REACH_VIDEOLATENCY_LIMIT event

Note: This feature is for MPEG-4, H.263, and H.264 channels.

When the total accumulated jitter buffer delay exceeds the allowed threshold for latency, the jitter buffer automatically sends an MSPEVN_REACH_VIDEOLATENCY_LIMIT unsolicited event to the application. The time-delay threshold for MSPEVN_REACH_VIDEOLATENCY_LIMIT events is 500 ms. Therefore, if the commanded latency is 300 ms, the jitter buffer generates an MSPEVN_REACH_VIDEOLATENCY_LIMIT event when the total delay in the jitter buffer is 800 ms.

MSPEVN_REACH_VIDEOLATENCY_LIMIT events notify the application that there may be enough delay in the jitter buffer to affect media synchronization. In response to these events, applications can:

- Monitor the overall delay with occasional queries to the jitter buffer using the MSP_QRY_JITTER_VIDEO_GET_STATE query.
- Remove excess packets with the MSP_CMD_JITTER_NORMALIZE_VIDEO_LATENCY_BUF command.
- Purge the jitter buffer with the MSP_CMD_JITTER_PURGE_VIDEO_LATENCY_BUF command.
- Take no action at all.

After the jitter buffer generates an MSPEVN_REACH_VIDEOLATENCY_LIMIT event, it cannot generate another MSPEVN_REACH_VIDEOLATENCY_LIMIT event until one of the following conditions occurs:

- Application issues a purge command for the jitter buffer
- Jitter buffer detects a source change in the incoming bit stream
- Jitter buffer encounters an overflow condition

Any of these conditions enables the jitter buffer to send another MSPEVN_REACH_VIDEOLATENCY_LIMIT event to the application when the delay threshold is exceeded.

Querying an H.264 endpoint for transmit status

Note: This feature is for full-duplex and simplex send H.264 video endpoints. The procedure and example shown are for full-duplex endpoints.

Query an H.264 video endpoint to determine H.264-specific information related to transmit packetization.

To query the H.264 endpoint, create the MSP_QRY_RTPFDX_H264_TX_STATUS query and send it to the CG board. The MSP_QRY_RTPFDX_H264_TX_STATUS query uses the msp_ENDPOINT_RTPFDX_H264_TX_STATUS structure to display parameters for the specified RTP endpoint.

For more information, see *Creating and sending MSPP queries* on page 47 and *msp_ENDPOINT_RTPFDX_H264_TX_STATUS* on page 95.

The following event can be returned:

Event	Description
MSPEVN_QUERY_DONE	MSP_QRY_RTPFDX_H264_TX_STATUS query was successfully sent to the specified endpoint on the CG board.

Example

The following example shows how to query the video jitter filter:

```
DWORD query;
query = mspBuildQuery(MSP_ENDPOINT_RTPFDX_VIDEO_H264, MSP_QRY_RTPFDX_H264_TX_STATUS);
if (mspSendQuery(ephd, query) != SUCCESS)
printf("\n\t ERROR: mspSendQuery failed.\n");
expectedEvent = (MSPEVN_QUERY_DONE | MSP_QRY_RTPFDX_H264_TX_STATUS);
if (WaitForSpecificEvent(gw, ctaQueueHd, expectedEvent, &pevent) != 0)
printf("\n\tERROR: mspSendQuery failed to send valid completion event\n");
printf("\n numPkts = %d, numNALs = %d, numAUs = %d, numIDRAUs = %d, numSPS = %d, numPPS =
%d, numSEI = %d",
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numPkts),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numNALs),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numAUs),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numIDRAUs),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numSPS),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numPPS),
NMS2H_DWORD(((msp_ENDPOINT_RTPFDX_H264_TX_STATUS*)(pevent->buffer))->numSEI));
mspReleaseBuffer(ctaQueueHd, pevent.buffer);
```

Configuring real-time timestamp generation

Note: This feature is for full-duplex and simplex video endpoints. The procedure and example are shown for full-duplex.

Some 3G-324M terminals can provide unreliable timing information in the video encoded bit streams they generate. This can lead to issues such as inaccurate timestamp information in RTP video packets, which can disrupt the synchronization of video packets in the media stream. In these cases, you can use the Video Access Real Time Timestamp (RTTS) feature as an alternative to generating RTP timestamps, so that reliable timestamp information is included in RTP packets.

Configure any video full-duplex or simplex send RTP endpoint to enable RTTS. When RTTS is enabled, Video Access uses a real-time clock, based on the arrival time of video frames from the 3G interface, to generate the RTP timestamps for transmitted RTP video packets. When RTTS is disabled (the default), Video Access generates RTP timestamps for video frames based on timing information received in the bit stream from the 3G peer video encoder.

To enable RTTS, create the MSP_CMD_RTPFDX_RTTS_CTRL command, and send it to the CG board.

The MSP_CMD_RTPFDX_RTTS_CTRL command uses the msp_ENDPOINT_RTPFDX_RTTS_CTRL structure to determine whether to enable or disable RTTS functionality. By default, RTTS is disabled.

For more information, see *Creating and sending MSPP commands* on page 47 and *msp_ENDPOINT_RTPFDX_RTTS_CTRL* on page 92.

The following event can be returned:

Event	Description
MSPEVN_SENDCOMMAND_DONE	MSP_CMD_RTPFDX_RTTS_CTRL command was successfully sent to the specified endpoint on the CG board.

Example

The following example shows how to enable RTTS for the endpoint with the MSP handle ephd:

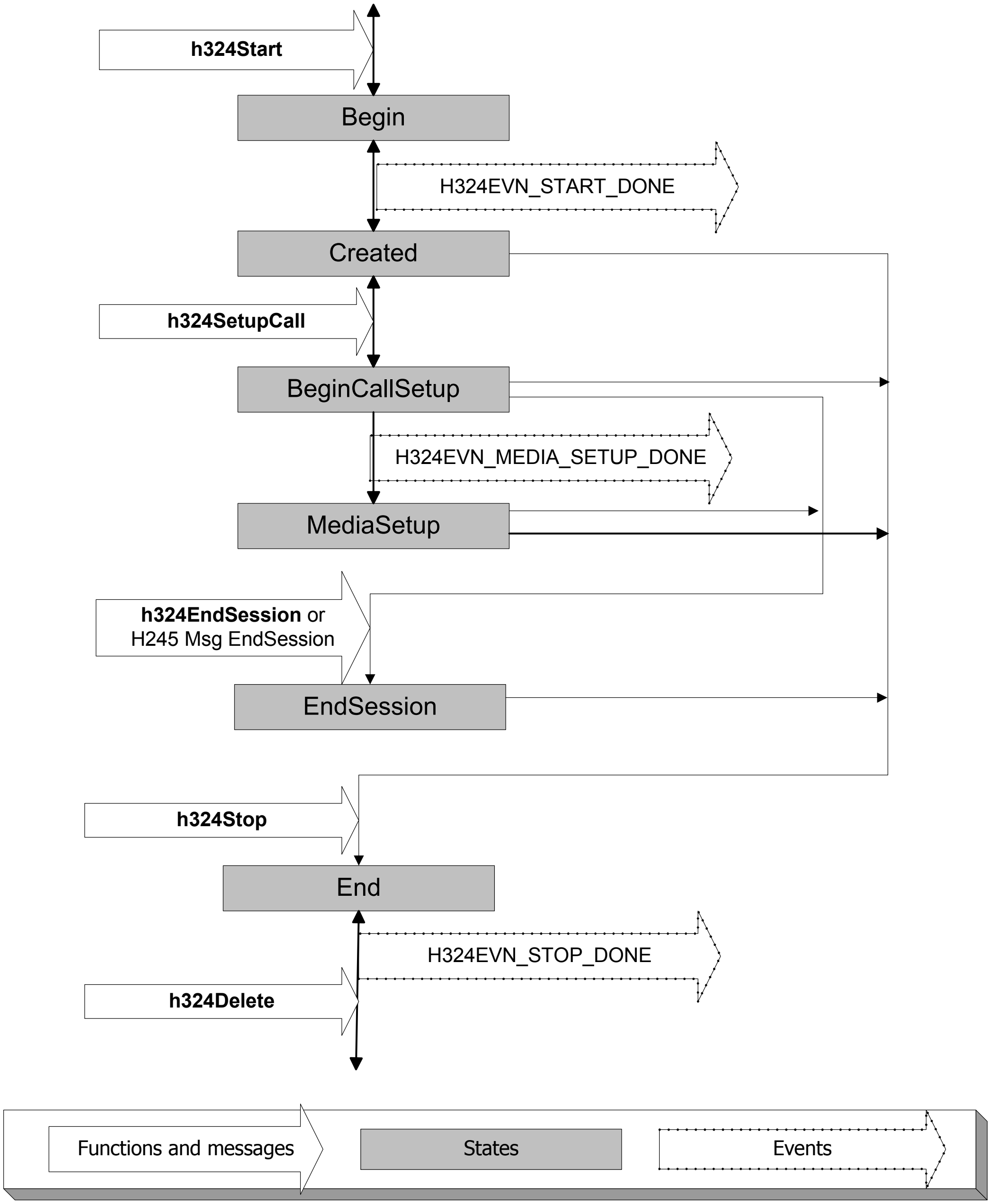
```
msp_ENDPOINT_RTPFDX_RTTS_CTRL rttsCmd;
rttsCmd.enable = H2NMS_DWORD(1);
command = mspBuildCommand(MSP_ENDPOINT_RTPFDX_VIDEO_H263,
              MSP_CMD_RTPFDX_RTTS_CTRL);
ret = mspSendCommand(ephd, command, &rttsCmd, sizeof(rttsCmd));
expectedEvent = (MSPEVN_SENDCOMMAND_DONE |
     MSP_CMD_RTPFDX_RTTS_CTRL);
if (WaitForSpecificEvent(gw,  CtaQueueHd, expectedEvent, &event) != 0)
printf("\n\tERROR: mspSendCommand failed to send valid completion event\n");
```

6 Call states

Call state sequence

During a call, each session of the H.324M Middleware passes through a sequence of states. The state transitions are driven by application functions, H.245 messages from the remote terminal, and internal actions of the 3G-324M Middleware.

The following illustration shows these state transitions:



Begin state

During the Begin state, the H.245 stack is instantiated, and structures are created and initialized.

Enters state	Upon receiving **h324Start**.
Exits state	After the H.245 stack is instantiated and structures are created and initialized, the Begin state transitions to the Created state. The state transition is signaled by H324EVN_START_DONE.
Related functions	None.
Normal events	H324EVN_START_DONE
Error events	H324EVN_H245_INTERNAL_ERROR

Created state

The Created state provides a holding point until the user is ready to begin call setup. During the Created state, TerminalCapabilitySet messages can be received from the remote terminal, but cannot be sent.

Enters state	Upon issuing H324EVN_START_DONE.
Exits state	One of the following: • Upon receiving **h324SetupCall**, the H.324M Middleware transitions to the BeginCallSetup state. • Upon receiving **h324EndSession** or the H.245 EndSession message, the H.324M Middleware transitions to the EndSession state. • Upon receiving **h324Stop**, the H.324M Middleware transitions to the End state.
Related functions	**h324EndSession** **h324GetTermCaps** **h324SetTermCaps** (but terminal capabilities are not sent until **h324SetupCall**) **h324SetupCall** **h324Stop**
Normal events	H324EVN_END_SESSION H324EVN_REMOTE_CAPABILITIES
Error events	H324EVN_CALL_SETUP_FAILED H324EVN_H245_INTERNAL_ERROR

BeginCallSetup state

The following steps occur during this state:

- Master slave determination is completed.
- Terminal capability set is sent and acknowledged.
- Terminal capability set is received and acknowledged.
- MUX tables exchange is completed.
- Logical channels negotiations are completed.

Enters state	Upon receiving **h324SetupCall**.
Exits state	One of the following: • After master slave determination is completed; the terminal capability set has been sent and acknowledged, and received and acknowledged; MUX tables have been exchanged; and both audio and video logical channels have been negotiated; the Middleware transitions to the MediaSetup state. The state transition is signaled by H324EVN_MEDIA_SETUP_DONE. • Upon receiving **h324EndSession** or receiving the H.245 EndSession message, the Middleware transitions to the EndSession state. • Upon receiving **h324Stop**, the Middleware transitions to the End state.
Related functions	**h324EndSession** **h324GetTermCaps** **h324SetTermCaps** **h324Stop**
Normal events	H324EVN_MEDIA_SETUP_DONE H324EVN_END_SESSION H324EVN_REMOTE_CAPABILITIES H324EVN_LCD
Error events	H324EVN_CALL_SETUP_FAILED H324EVN_H245_INTERNAL_ERROR

MediaSetup state

Media channels are open and the call is in progress.

Enters state	Upon issuing H324EVN_MEDIA_SETUP_DONE.
Exits state	One of the following: • Upon receiving **h324EndSession** or receiving the H.245 EndSession message, the H.324M Middleware transitions to the EndSession state. • Upon receiving **h324Stop**, the H.324M Middleware transitions to the End state.
Related functions	**h324EndSession** **h324RoundTripDelay** **h324UserIndication** **h324VideoFastUpdate** **h324Stop**
Normal events	H324EVN_END_SESSION H324EVN_ROUND_TRIP_DELAY H324EVN_USER_INDICATION H324EVN_VIDEO_FAST_UPDATE
Error events	H324EVN_H245_INTERNAL_ERROR H324EVN_ROUND_TRIP_TIMEOUT

EndSession state

Sends out the H.245 EndSession message or processes an incoming H.245 EndSession message, and then waits to be shut down.

Enters state	Upon receiving **h324EndSession** or receiving the H.245 EndSession message.
Exits state	Upon receiving **h324Stop**, the Middleware transitions to the End state.
Related functions	**h324Stop**
Normal events	None.
Error events	H324EVN_H245_INTERNAL_ERROR

End state

Shuts down the H.245 stack.

Enters state	Upon receiving **h324Stop**.
Exits state	When the H.245 stack has been shut down, the state is destroyed. This is signaled by H324EVN_STOP_DONE.
Related functions	None.
Normal events	H324EVN_STOP_DONE
Error events	H324EVN_H245_INTERNAL_ERROR

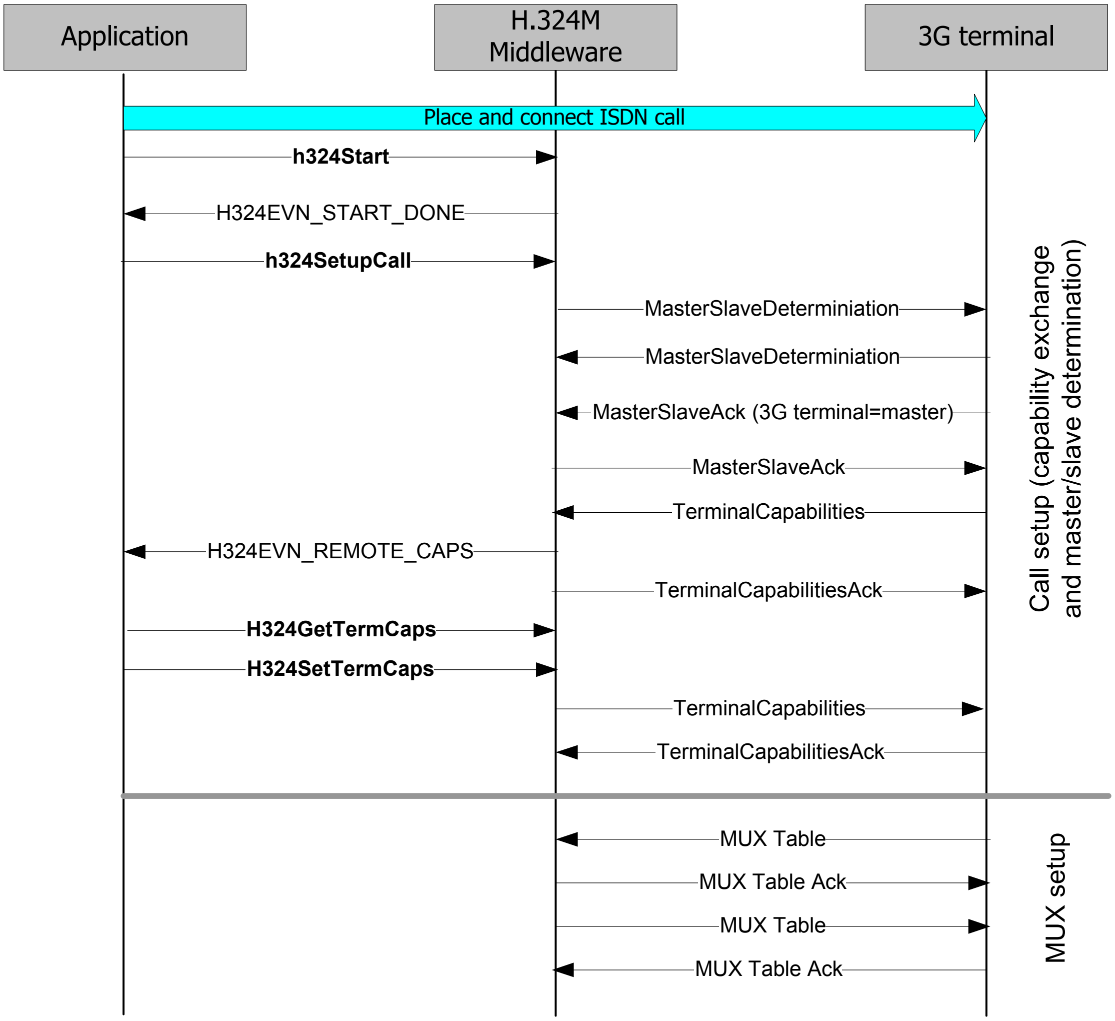
7 Call flow examples

Example 1: Negotiating video with AL3

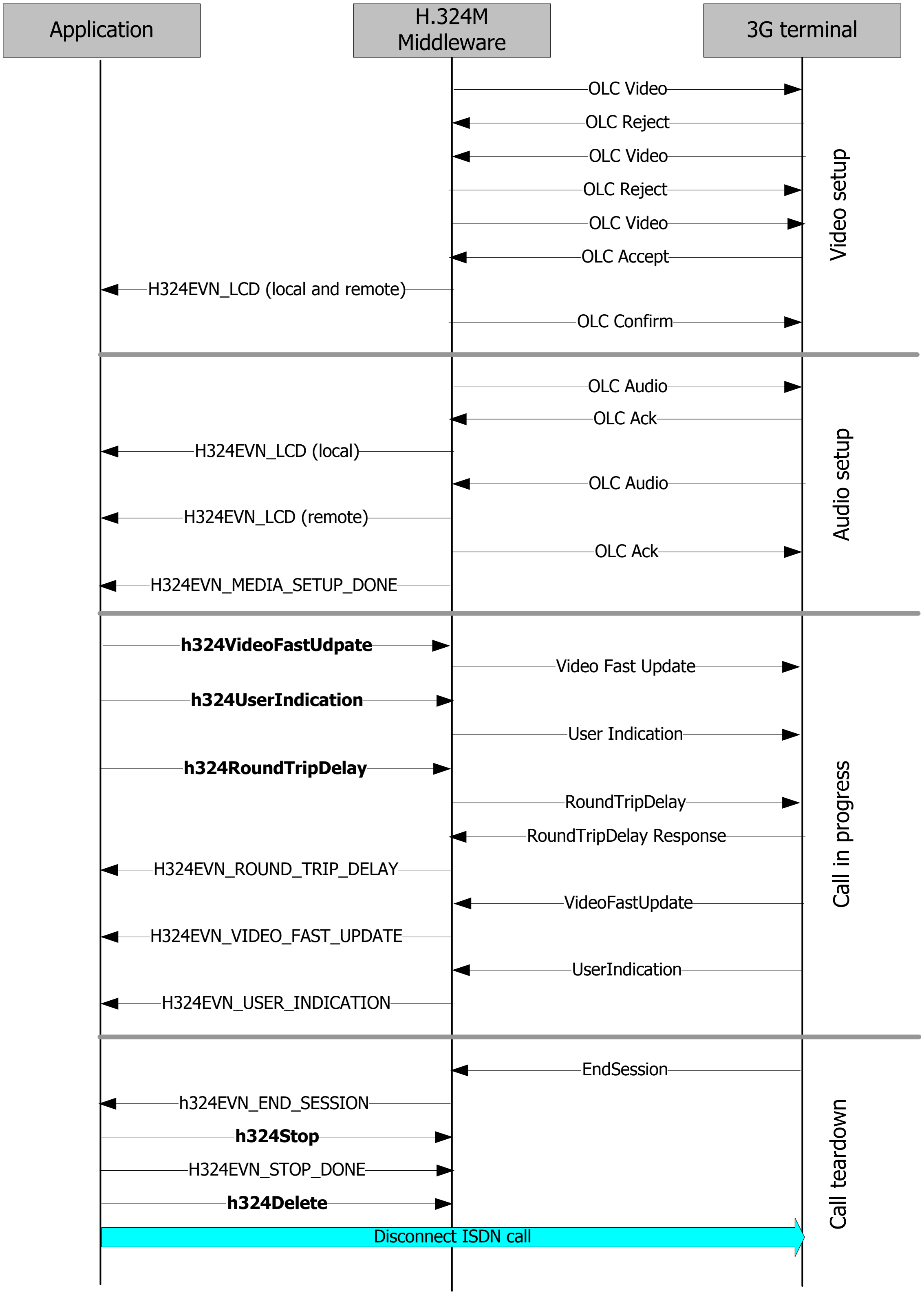
This topic shows an example where the application initiates the call, and both sides negotiate video with AL3 using bi-directional video OLCs. (Either one or both sides do not support AL2.) It describes this call flow in two parts:

- Part 1 describes the communication among the application, H.324M Middleware, and 3G terminal for master/slave determination and for setting up the MUX.
- Part 2 describes the communication among the application, H.324M Middleware, and 3G terminal for setting up video and audio. It also describes the communication for a call in progress and for call teardown.

Negotiating video with AL3 – Part 1



Negotiating video with AL3 – Part 2

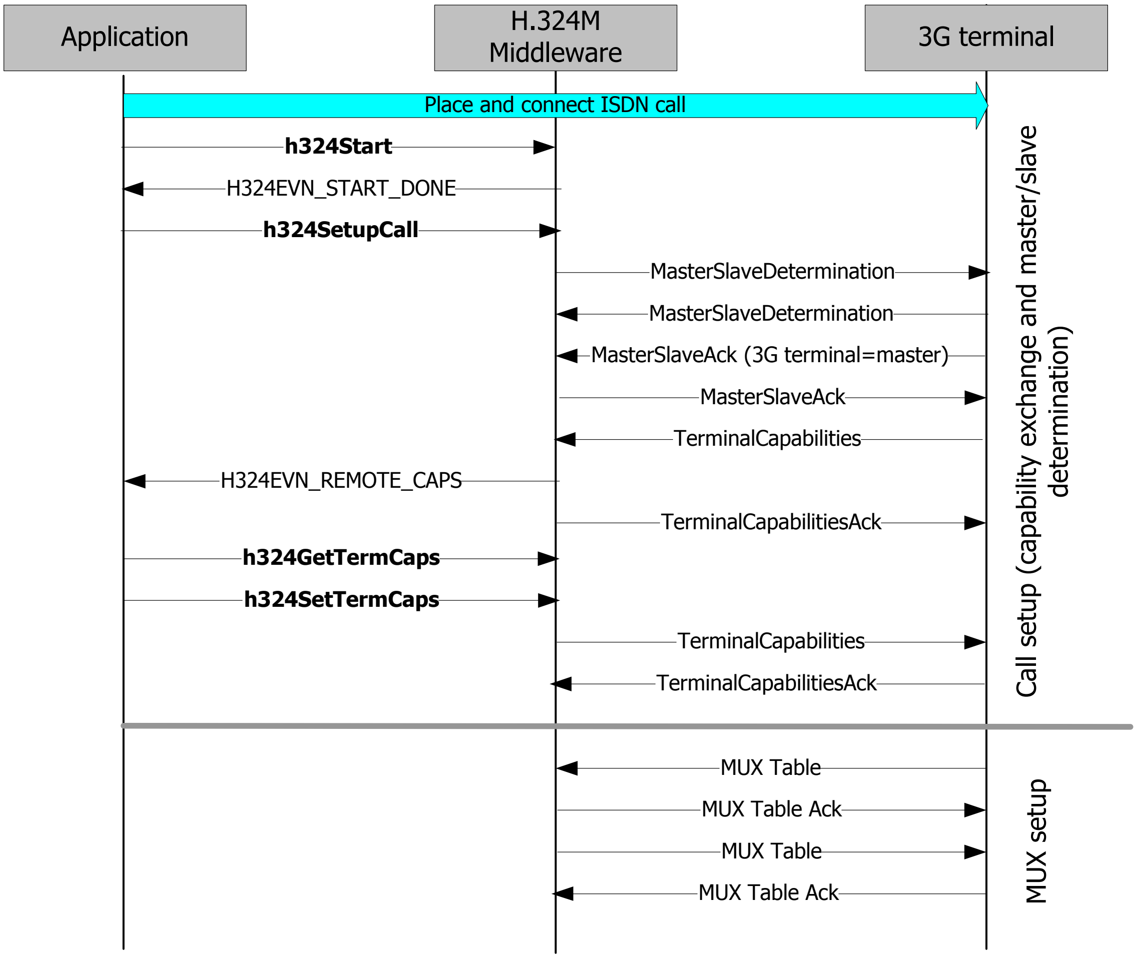


Example 2: Negotiating video with AL2

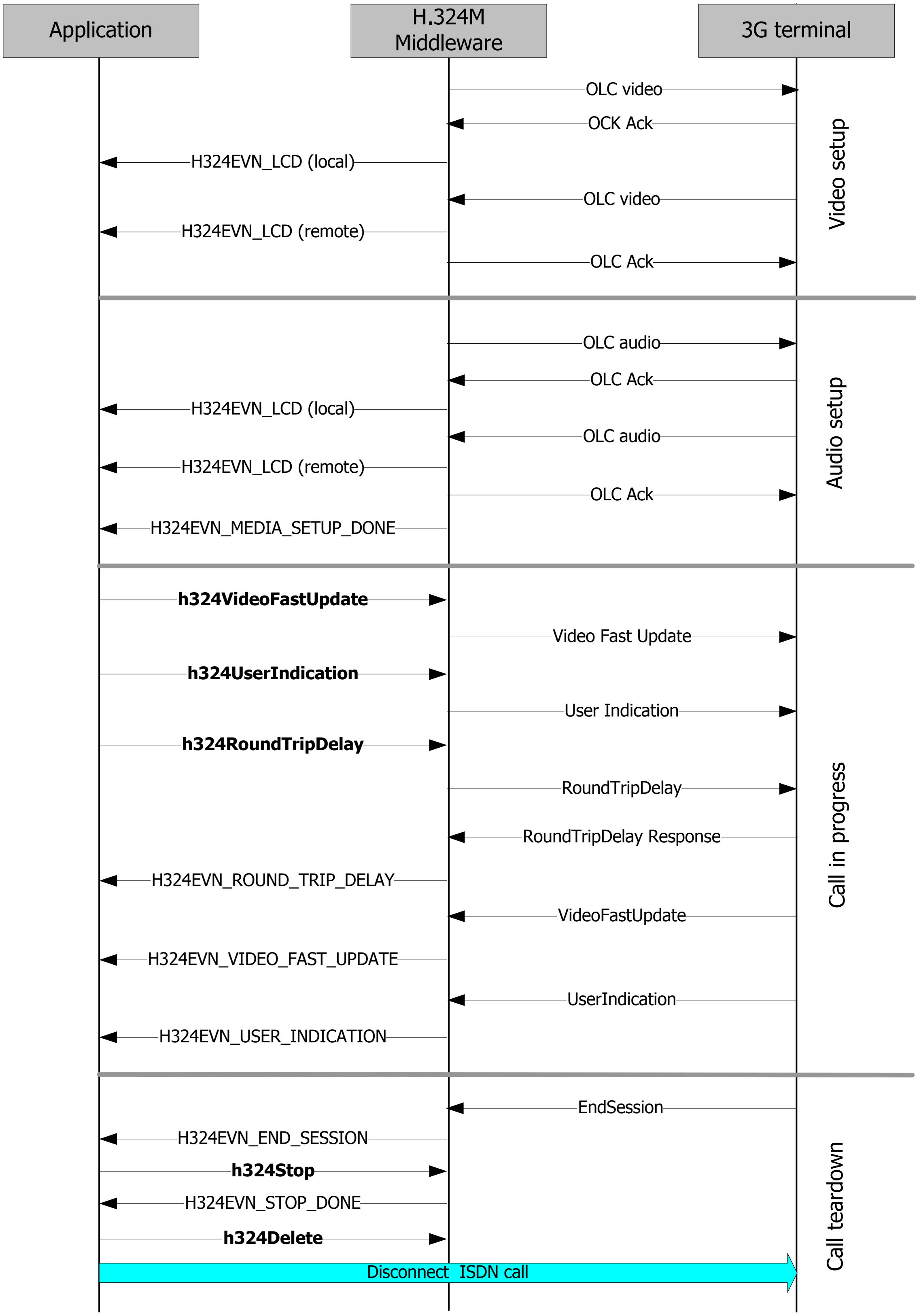
This topic shows an example where the application initiates the call, and both sides negotiate video with AL2 using unidirectional video OLCs. (Both sides support video AL2.) It describes this call flow in two parts:

- Part 1 describes the communication among the application, H.324M Middleware, and 3G terminal for master/slave determination and for setting up the MUX.
- Part 2 describes the communication among the application, H.324M Middleware, and 3G terminal for setting up video and audio. It also describes the communication for a call in progress and for call teardown.

Negotiating video with AL2 – Part 1



Negotiating video with AL2 – Part 2

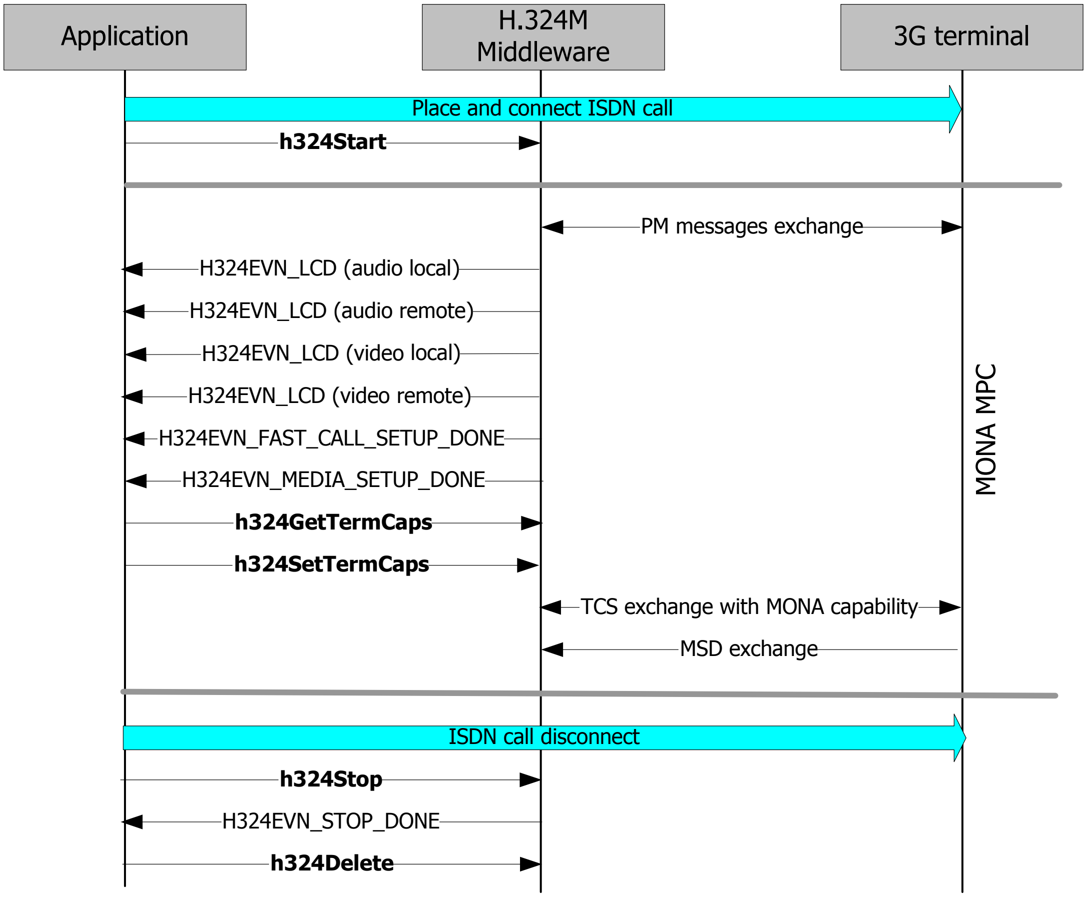


Example 3: Negotiating fast call setup using MONA

This topic shows two examples of negotiating fast call setup using MONA.

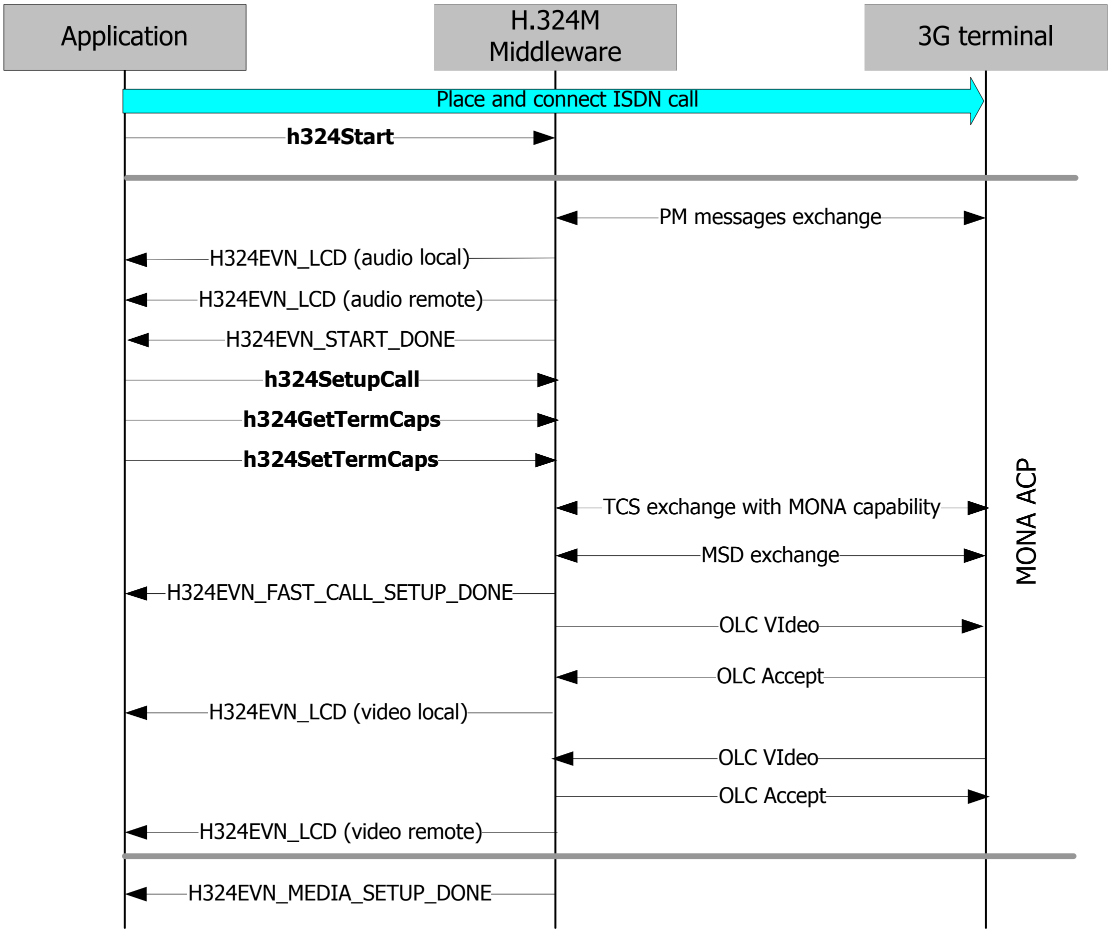
Using the media preconfigured channels (MPC) procedure

In this example, two endpoints negotiate successfully using the MONA media preconfigured channels (MPC) procedure. For more information, see *MONA* on page 24.



Falling back to accelerated H.245 signaling

In this example, MPC cannot set up the video channels because of non-matching video codecs, but the two endpoints negotiate successfully using the MONA accelerated H.245 signaling technique (ACP). For more information, see *Enabling fast call setup* on page 23.

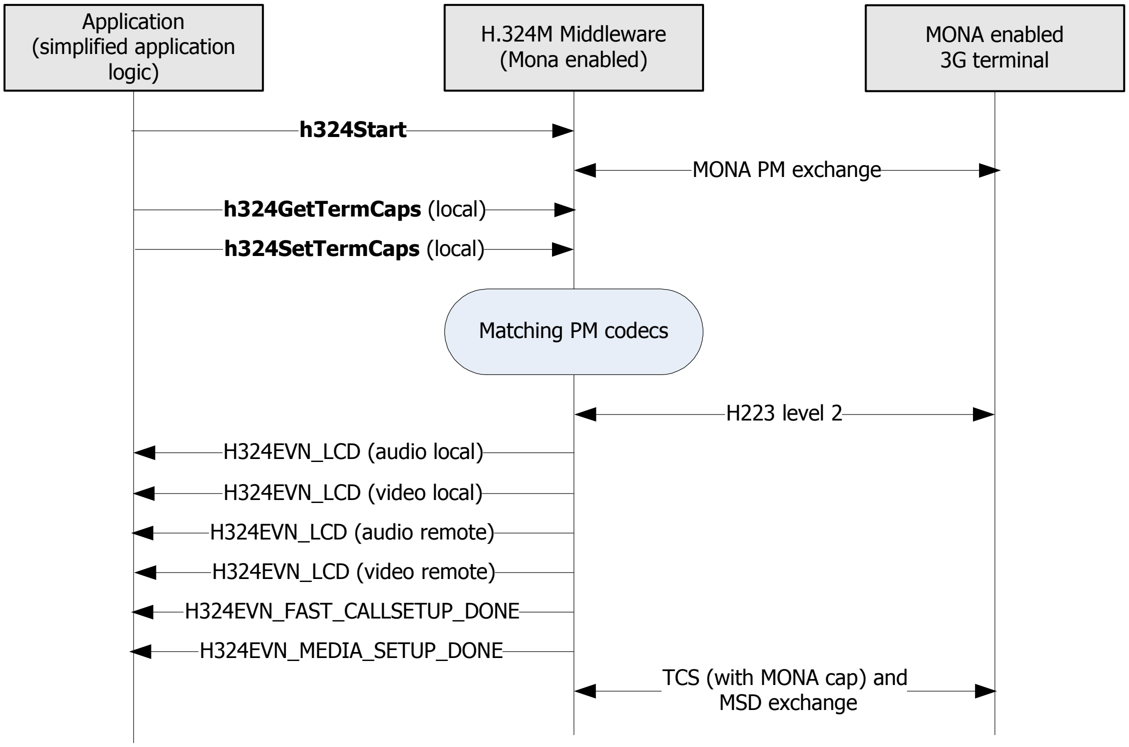


Example 4: Using simplified application logic to set up calls

This topic shows three examples of negotiating call setup using simplified application logic: the application sets the local capabilities after calling **h324Start**.

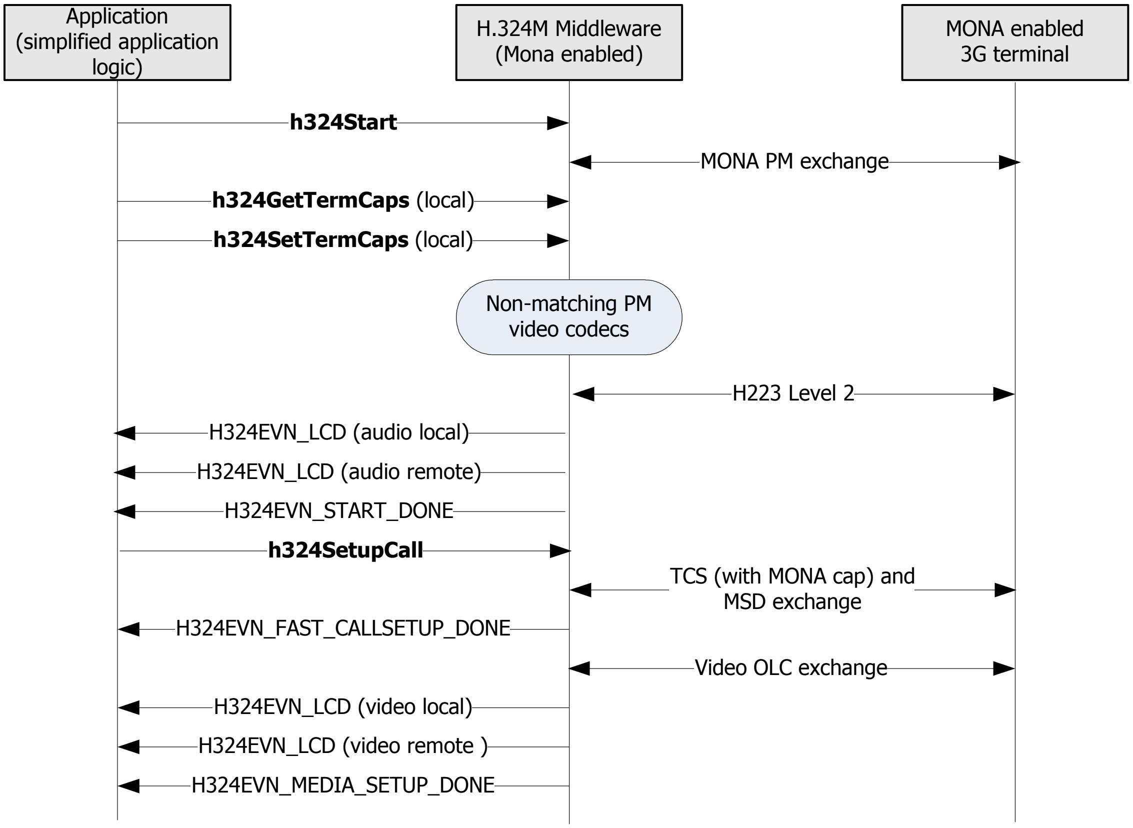
MONA MPC call setup

In this example, two endpoints successfully negotiate using the preconfigured MONA media:


Application (simplified application logic)
H.324M Middleware (Mona enabled)
MONA enabled 3G terminal
h324Start
MONA PM exchange
h324GetTermCaps (local)
h324SetTermCaps (local)
Matching PM codecs
H223 level 2
H324EVN_LCD (audio local)
H324EVN_LCD (video local)
H324EVN_LCD (audio remote)
H324EVN_LCD (video remote)
H324EVN_FAST_CALLSETUP_DONE
H324EVN_MEDIA_SETUP_DONE
TCS (with MONA cap) and MSD exchange

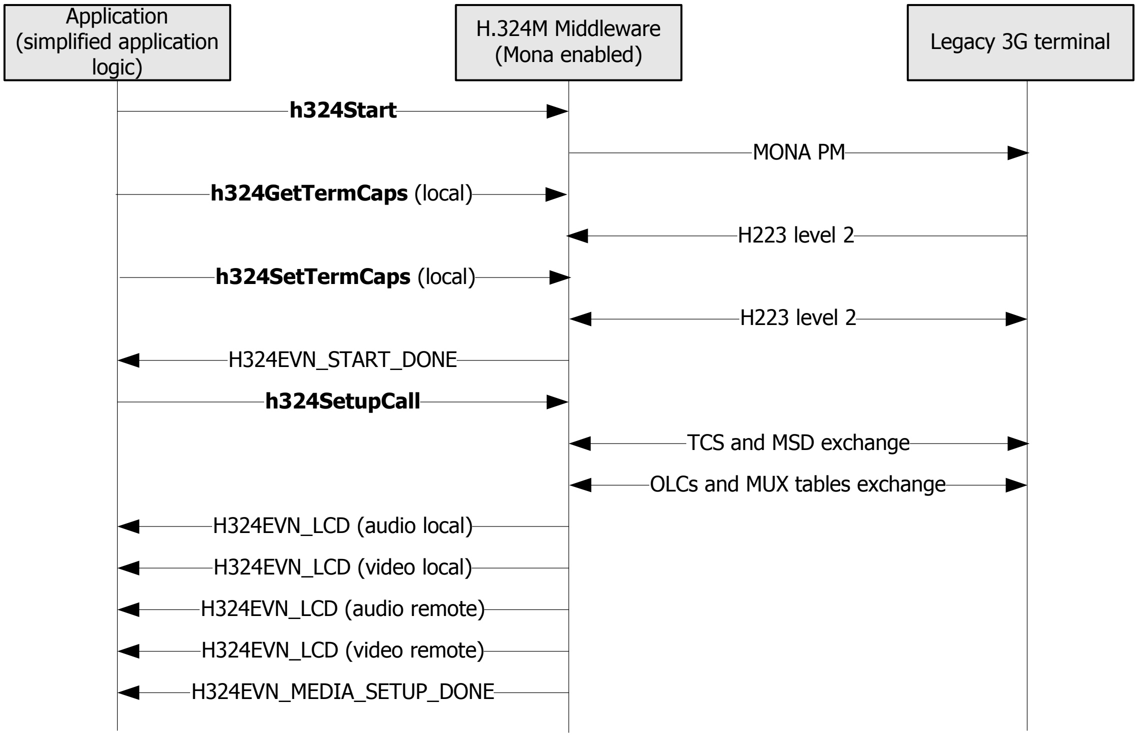
MONA ACP fallback call setup

In this example, MPC cannot be used to set up the video channels because of non-matching video codecs:


Application (simplified application logic)
H.324M Middleware (Mona enabled)
MONA enabled 3G terminal
h324Start
MONA PM exchange
h324GetTermCaps (local)
h324SetTermCaps (local)
Non-matching PM video codecs
H223 Level 2
H324EVN_LCD (audio local)
H324EVN_LCD (audio remote)
H324EVN_START_DONE
h324SetupCall
TCS (with MONA cap) and MSD exchange
H324EVN_FAST_CALLSETUP_DONE
Video OLC exchange
H324EVN_LCD (video local)
H324EVN_LCD (video remote)
H324EVN_MEDIA_SETUP_DONE

Falling back to standard H.245 signaling

In this example, the remote terminal does not support MONA:


Application (simplified application logic)
H.324M Middleware (Mona enabled)
Legacy 3G terminal
h324Start
MONA PM
h324GetTermCaps (local)
H223 level 2
h324SetTermCaps (local)
H223 level 2
H324EVN_START_DONE
h324SetupCall
TCS and MSD exchange
OLCs and MUX tables exchange
H324EVN_LCD (audio local)
H324EVN_LCD (video local)
H324EVN_LCD (audio remote)
H324EVN_LCD (video remote)
H324EVN_MEDIA_SETUP_DONE

8 MSPP video-enhanced structures

MSPP video-enhanced structure categories

This topic categorizes the MSPP video-enhanced structures. Subsequent topics describe each structure in alphabetical order.

Structures for creating MSPP endpoints

Use the following MSPP video-enhanced structures to create MSPP endpoints:

- MSP_ENDPOINT_ADDR
- MSP_ENDPOINT_PARAMETER
- MUX_ENDPOINT_ADDR
- RTP_PAYLOAD_MAP
- RTPRTCP_ENDPOINT_ADDR
- RTPRTCP_ENDPOINT_PARMS
- RTPRTCP_V6_ENDPOINT_ADDR
- RTPRTCP_V6_ENDPOINT_PARMS

Structures for creating MSPP channels

Use the following MSPP video-enhanced structures to create MSPP channels:

- MSP_AUDIO_CHANNEL_PARMS
- MSP_CHANNEL_ADDR
- MSP_CHANNEL_PARAMETER
- MSP_VIDEO_CHANNEL_PARMS

Structures for modifying the 3G-324M Interface configuration

Use the following video-enhanced structures to modify the 3G-324M Interface configuration:

- msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC
- msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL
- msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL
- msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI
- msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL
- msp_ENDPOINT_RTPFDX_SET_VID_SKEW
- msp_ENDPOINT_RTPFDX_SET_VID_TX_PID
- msp_FILTER_JITTER_VID_LATENCY

Structure for querying the video jitter buffer

Use the msp_FILTER_JITTER_VIDEO_STATE structure to get the status of the video jitter buffer.

Structure for querying the H.264 video endpoint

Use the msp_ENDPOINT_RTPFDX_H264_TX_STATUS structure to get the transmit status of an H.264 video endpoint.

MSP_AUDIO_CHANNEL_PARMS

Defines the MSPP channel used for AMR transcoding in the 3G-324M Interface. MSP_AUDIO_CHANNEL_PARMS is used as a parameter in the MSP_CHANNEL_PARAMETER structure, which is used by **mspCreateChannel**.

```
typedef struct tag_MSP_AUDIO_CHANNEL_PARMS
{
    DWORD                       size;
    msp_FILTER_ENCODER_PARMS    EncoderParms;
    msp_FILTER_DECODER_PARMS    DecoderParms;
} MSP_AUDIO_CHANNEL_PARMS;
```

For more information, see *Designing the 3G-324M Interface channel configurations* on page 12, *Creating a channel* on page 34 and the *MSPP Service Developer's Reference Manual*.

MSP_CHANNEL_ADDR

Creates audio and video MSPP channels for the 3G-324M Interface. MSP_CHANNEL_ADDR is used by **mspCreateChannel**.

When you use this structure, you must enter a channel type identifier in the channelType field. See *Creating a channel* on page 34 for a list of channel type identifiers. You can enable inband DTMF carriage capability for an audio channel by setting the FilterAttribs field to MSP_FCN_ATTRIB_RFC2833.

```
typedef struct tag_MSP_CHANNEL_ADDR
{
    DWORD               size;
    DWORD               nBoard;        // Channel location (board number)
    MSP_CHANNEL_TYPE    channelType;
    DWORD               FilterAttribs; // Used to enable DSP Filter functions
} MSP_CHANNEL_ADDR;
```

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

MSP_CHANNEL_PARAMETER

Sets MSPP channel parameters for audio and video channels. MSP_CHANNEL_PARAMETER is used by **mspCreateChannel**.

The parameters specified in the MSP_CHANNEL_PARMETER structure depend on the type of channel created. For example, if you create a video channel, use the MSP_VIDEO_CHANNEL_PARMS structure as a parameter for MSP_CHANNEL_PARAMETER. See *Creating a channel* on page 34 for a list of the structures associated with specific endpoint types.

```
typedef struct tag_MSP_CHANNEL_PARAMETER
{
    DWORD               size;
    MSP_CHANNEL_TYPE    channelType;

union
    {
        MSP_VOICE_CHANNEL_PARMS            VoiceParms;
        MSP_FAX_CHANNEL_PARMS              FaxParms;
        MSP_RTP_SWITCHING_CHANNEL_PARMS    RtpSwitchingParms;
        MSP_VIDEO_CHANNEL_PARMS            VideoParms;
        MSP_AUDIO_CHANNEL_PARMS            AudioParms;
        MSP_UNDEFINED_CHANNEL_PARMS        Undefined;
        // Structure may be expanded to define new Channel Types
    } ChannelParms;

} MSP_CHANNEL_PARAMETER;
```

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

MSP_ENDPOINT_ADDR

Creates MUX endpoints, audio endpoints, and video endpoints for the 3G-324M Interface. MSP_ENDPOINT_ADDR is used by **mspCreateEndpoint**.

The parameters specified in the MSP_ENDPOINT_ADDR structure depend on the type of endpoint created. For example, if you create an MPEG-4, H.263, or H.264 video endpoint for IPv4, use the RTPRTCP_ENDPOINT_ADDR structure as a parameter for MSP_ENDPOINT_ADDR. If you create a video endpoint for IPv6, use the RTPRTCP_V6_ENDPOINT_ADDR structure as a parameter for MSP_ENDPOINT_ADDT. See *Creating an endpoint* on page 31 for a list of the structures associated with specific endpoint types.

```
typedef struct tag_MSP_ENDPOINT_ADDR
{
    DWORD   size;       // ENDPOINT_ADDR
    DWORD   nBoard;     // board number
    DWORD   eEpType;    // MSP_ENDPOINT_DS0, MSP_ENDPOINT_RTPFDX, etc

union
    {
        DS0_ENDPOINT_ADDR           DS0;
        PKTMEDIA_ENDPOINT_ADDR      Pktmedia;
        MONITOR_ENDPOINT_ADDR       Monitor;
        RTPRTCP_ENDPOINT_ADDR       RtpRtcp;
        UDP_ENDPOINT_ADDR           Udp;
        T38UDP_ENDPOINT_ADDR        T38Udp;
        MUX_ENDPOINT_ADDR           Mux;

        // New endpoints to support IPv6
        RTPRTCP_V6_ENDPOINT_ADDR    RtpRtcpV6
        UPD_V6_ENDPOINT_ADDR        UdpV6

        // Structure may be expanded to define new Endpoints
    } EP;

} MSP_ENDPOINT_ADDR;
```

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

MSP_ENDPOINT_PARAMETER

Sets configuration parameters for MUX endpoints, audio endpoints, and video endpoints for the 3G-324M Interface. The MSP_ENDPOINT_PARAMETER structure is used by **mspCreateEndpoint**.

The parameters specified in the MSP_ENDPOINT_PARAMETER structure depend on the type of endpoint you create. For example, if you create an MPEG-4, H.263, or H.264 video endpoint video endpoint for IPv4, use the RTPRTCP_ENDPOINT_PARMS structure as a parameter for MSP_ENDPOINT_PARAMETER. If you create a video endpoint for IPv6, use the RTPRTCP_V6_ENDPOINT_PARMS structure as a parameter for MSP_ENDPOINT_PARAMETER. See *Creating an endpoint* on page 31 for a list of the structures associated with specific endpoint types.

```
typedef struct tag_MSP_ENDPOINT_PARAMETER
{
    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 information about **mspCreateEndpoint**, see the *MSPP Service Developer's Reference Manual*.

msp_ENDPOINT_RTPFDX_ENABLE_SKEW_CALC

This video-specific structure enables or disables 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.

Use this structure with the MSP_CMD_RTPFDX_CALC_SKEW_OFFSET command, which is described in *Using RTCP for audio/video synchronization* on page 58.

Definition

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

Field

Field name	Default	Description
enable	0	Controls skew value calculations. Valid values are: 0 in 4th bit - Disables skew calculation. 1 in 4th bit - Enables skew calculation. Use the RTCP_ENABLE_RCV_SKEW_CALC macro to set the appropriate bit. For more information, see *RTPRTCP_ENDPOINT_PARMS* on page 97.

msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL

This video-specific structure sets the RFC encapsulation type used for H.263 endpoints. RFC 2190 is the default encapsulation type.

Use this structure with the MSP_CMD_RTPFDX_H263_ENCAP_CTRL command, which is described in *Setting the H.263 RFC encapsulation type* on page 55.

Definition

```
typedef struct {
    #define   MSP_H263_RFC2190 0  /* Encapsulate H.263 using RFC 2190 payload headers */
    #define   MSP_H263_RFC2429 1  /* Encapsulate H.263+ using RFC 2429 payload header */
    U32       h263Encap;          // MSP_H263_RFC2190 or MSP_H263_RFC2429
} msp_ENDPOINT_RTPFDX_H263_ENCAP_CTRL;
```

Fields

Field name	Description
h263Encap	Type of encapsulation. Valid values are: MSP_H263_RFC2190 - (default) Uses and expects RFC 2190 payload headers. MSP_H263_RFC2429 - Uses and expects RFC 2429 payload headers.

msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL

This video-specific structure enables or disables I-frame notifications for all types of video endpoints.

Use msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL with the MSP_CMD_RTPFDX_IFRAME_NOTIFY_CTRL command, which is described in *Enabling I-frame notification* on page 49.

Definition

```
typedef struct
{
    U32    h324Notify;    // Freq. to send msg when I-Frame arrives on H.324M
    U32    ipNotify;      // Freq. to send msg when I-Frame arrives on RTP/IP
} msp_ENDPOINT_RTPFDX_IFRAME_NOTIFY_CTRL;
```

Fields

Field name	Default	Description
h324Notify	0	Controls the notification of I-frames arriving from the PSTN side of the 3G-324M Interface. Valid values are: 0 - Disables the sending of I-frame notifications for this direction on the endpoint. 1 - Sends a notification for the next arriving I-frame for this direction on the endpoint. 2 - Sends a notification for *each* arriving I-frame for this direction on the endpoint.
ipNotify	0	Controls the notification of I-frames arriving from the IP side of the 3G-324M Interface. Valid values are: 0 - Disables the sending of I-frame notifications for this direction on the endpoint. 1 - Sends a notification for the next arriving I-frame for this direction on the endpoint. 2 - Sends a notification for *each* arriving I-frame for this direction on the endpoint.

msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI

This video-specific structure sends out-of-band DCI for MPEG-4 and H.264 endpoints.

Use msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI with the MSP_CMD_RTPFDX_OUT_OF_BAND_DCI command, which is described in *Inserting out-of-band DCI into the video bit stream* on page 56.

Definition

```
typedef struct
{
    DWORD len;                     // Length of DCI
    U8 mode;                       // 0: Mode for how to use DCI
    U8 data[MAX_DCI_BUF_SIZE];     // Buffer for saving the out_of_band_dci data.
} msp_ENDPOINT_RTPFDX_OUT_OF_BAND_DCI;
```

Fields

Field name	**Default**	**Description**
len	N/A	Length of the out-of-band DCI, in bytes. Range is 0 to 255 bytes.
mode	0	Indicates what in-band DCI to replace with out-of-band DCI. Valid values are: 0 - No change. 1 - Uses out-of-band DCI to replace only the initial DCI at the beginning of the bit stream. 2 - Replaces all in-band DCI with the one received out-of-band. 3 - Same as value 2, but additionally inserts one out-of-band DCI before every I-frame.
data	N/A	Out-of-band DCI to be sent to the RTP endpoint. For MPEG-4, the data stream is raw VO/VOL data. For H.264, the data stream is sets of SPS/PPS NAL units separated by start codes per the byte stream format of Annex B of H.264.

msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL

This video-specific structure fine-tunes CG board performance by controlling the RTP packetization parameters transmitted by MPEG-4, H.263, and H.264 RTP endpoints from the PSTN side to the IP side of the 3G-324M Interface.

Use msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL with the MSP_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL command and the MSP_QRY_RTPFDX_VIDEO_RTP_PKTSZ_CTRL query, which are described in *Adjusting RTP packetization parameters* on page 52.

Note: NMS recommends that you keep the default values for this structure, unless you must fine-tune performance for a video tromboning configuration.

Definition

```
typedef struct tag_msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL
{
    DWORD    pktMaxSz;
    DWORD    aggThreshold;
    DWORD    enableAggregation;
} msp_ENDPOINT_RTPFDX_RTP_PKTSZ_CTRL;
```

Fields

Field name	Default	Description
pktMaxSz	1400	Sets the maximum transmission size in bytes for the payload of RTP packets being transmitted from the specified video RTP endpoint. Valid values are: 0 - Uses the default size (1400). -1 - Leaves the current value of pktMaxSz unmodified. A value between MIN_PKTMAXSIZE and 1440, inclusive.
aggThreshold	700	Sets the aggregation threshold in bytes for the transmission of aggregated packets. This parameter is not used for H.264 endpoints. Valid values are: 0 - Uses the default threshold (700). -1 - Leaves the current value of aggThreshold unmodified. A value between MIN_AGGTHRESHOLD and 1440, inclusive.
enableAggregation	Enabled	Indicates whether the video RTP endpoint aggregates H.263 GOB frames (for H.263 endpoints) or MPEG-4 Video Packets (for MPEG-4 endpoints) before transmitting them. This parameter is not used for H.264 endpoints. If aggregation is enabled, then the RTP endpoint aggregates consecutive GOB/Video Packet frames, until adding another GOB/Video Packet frame causes the outgoing RTP packet to exceed the bytes specified in the aggThreshold field. Valid values are: 0 - Disables the aggregation of GOB/Video Packet frames. -1 - Leaves the current value of the enableAggregation field unmodified. A value other than -1 or 0 - Enables the aggregation.

msp_ENDPOINT_RTPFDX_RTTS_CTRL

This video-specific structure enables or disables RTTS for MPEG-4, H.263, and H.264 endpoints.

Use msp_ENDPOINT_RTPFDX_RTTS_CTRL with the MSP_CMD_RTPFDX_RTTS_CTRL command, which is described in Configuring real-time timestamp generation.

Definition

```
typedef struct
{
U32         enable;                       // NZ=enable, Z=disable
} msp_ENDPOINT_RTPFDX_RTTS_CTRL;
```

Fields

Field name	Default	Description
enable	0	Enables or disables the RTTS feature. Valid values are: 0 - Disables RTTS. Non-zero value – Enables RTTS.

msp_ENDPOINT_RTPFDX_SET_VID_SKEW

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

Use this structure with the MSP_CMD_RTPFDX_VIDEO_SKEW_TIME command, which is described in *Enabling RTP video send endpoints to send video skew values to the IP destination* on page 61.

Definition

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

Fields

Field name	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. For more information, see *Using RTCP for audio/video synchronization* on page 58.

msp_ENDPOINT_RTPFDX_SET_VID_TX_PID

This video-specific structure sets the transmit RTP payload ID for video full-duplex and simplex send RTP endpoints.

Use this structure with the MSP_CMD_RTPFDX_SET_VID_TX_PID command, which is described in *Setting the video transmit RTP payload ID* on page 57.

Definition

```
typedef struct
{
    U32  txPayloadId;          //Transmit Payload ID, max = 127
} msp_ENDPOINT_RTPFDX_SET_VID_TX_PID;
```

Fields

Field name	Default	Description
txPayloadId	MPEG-4 - 100 H.263/2190 - 34 H.263/2429 - 101 H.264 - 104	Transmit payload ID. Valid values are 0 - 127.

msp_FILTER_JITTER_VID_LATENCY

This video-specific structure sets a video jitter buffer latency value.

Use this structure with the MSP_CMD_JITTER_CHG_VIDEO_LATENCY command, which is described in *Setting the video jitter buffer latency* on page 62.

Definition

```
typedef struct tag_msp_JITTER_VID_LATENCY
{
    DWORD   value;
} msp_FILTER_JITTER_VID_LATENCY;
```

Fields

Field name	Default	Description
value	0	Video jitter buffer latency duration in ms. Valid values are 0 - 2000.

msp_FILTER_JITTER_VIDEO_STATE

This video-specific structure gets the status of the video jitter buffer.

Use this structure with the MSP_QRY_JITTER_VIDEO_GET_STATE query, which is described in *Querying the video jitter buffer state* on page 63.

Definition

```
typedef struct tag_msp_FILTER_JITTER_VIDEO_STATE {
    DWORD   FilterId;
    DWORD   pktsReceived;
    DWORD   pktsAccepted;
    DWORD   pktsRejected;
    DWORD   pktsLost;
    DWORD   pktsCurrent;
    DWORD   overflows;
    DWORD   underflows;
    DWORD   duplicates;
    DWORD   lates;
    DWORD   reorders;
    DWORD   maxBuffers;
    DWORD   jitterBufDelay;
    DWORD   configuredLatency;
    DWORD   h264AggPkts;
    DWORD   h264FragPkts;
    DWORD   pktsDiscarded;
}msp_FILTER_JITTER_VIDEO_STATE;
```

Fields

Field name	Description
FilterId	Video jitter filter identification.
pktsReceived	Number of packets received in the video jitter buffer.
pktsAccepted	Number of packets accepted by the video jitter buffer.
pktsRejected	Number of packets rejected by the video jitter buffer.
pktsLost	Number of packets lost.
pktsCurrent	Number of packets currently in the video jitter buffer.
overflows	Number of times the video jitter buffer completely filled with packets.
underflows	Number of times the video jitter buffer was empty.
duplicates	Number of duplicate packets that arrived at the video jitter buffer.
lates	Number of times that packets came out of order.
reorders	Number of times that packets were reordered due to late packets.
maxBuffers	Maximum number of video jitter buffers used.
jitterBufDelay	Duration of data accumulated in the video jitter buffer in ms.
configuredLatency	Configured video jitter buffer latency in ms.
h264AggPkts	Number of H.264 aggregation packets received (H.264 only).
h264FragPkts	Number of H.264 fragmentation packets received (H.264 only).
pktsDiscarded	Number of packets discarded after initially being accepted (for example, due to overflow).

msp_ENDPOINT_RTPFDX_H264_TX_STATUS

This video-specific structure gets the transmit status of an H.264 full-duplex or simplex send video endpoint.

Use this structure with the MSP_QRY_RTPFDX_H264_TX_STATUS query, which is described in *Querying an H.264 Endpoint for Transmit Status* on page 67.

Definition

```
typedef struct tag_msp_ENDPOINT_RTPFDX_H264_TX_STATUS {
DWORD   FilterId;
DWORD   numPkts;
DWORD   numNALs;
DWORD   numAUs;
DWORD   numIDRAUs;
DWORD   numSPS;
DWORD   numPPS;
DWORD   numSEI;
} msp_ENDPOINT_RTPFDX_H264_TX_STATUS;
```

Fields

Field name	Description
FilterId	H.264 endpoint filter identification.
numPkts	Number of packets transmitted.
numNALs	Number of NAL units detected from 324M.
numAUs	Number of access units transmitted.
numIDRAUs	Number of IDR access units transmitted.
numSPS	Number of SPS NAL units received from 324M.
numPPS	Number of PPS NAL units received from 324M.
numSEI	Number of SEI NAL units received from 324M.

MSP_VIDEO_CHANNEL_PARMS

This video-specific structure defines the jitter buffer parameters for a video channel. These parameters cannot be changed.

Use MSP_VIDEO_CHANNEL_PARMS as a parameter in the MSP_CHANNEL_PARAMETER structure, which is used with **mspCreateChannel**.

```
typedef struct tag_MSP_VIDEO_CHANNEL_PARMS
{
    DWORD size;
    msp_FILTER_JITTER_PARMS JitterParms; // Video Jitter parameters cannot
                                         // be modified
} MSP_VIDEO_CHANNEL_PARMS;
```

For more information, see *Creating a channel* on page 34 and the *MSPP Service Developer's Reference Manual*.

MUX_ENDPOINT_ADDR

This video-specific structure uses the timeslot of a MUX endpoint. MUX_ENDPOINT_ADDR is used as a parameter in the MSP_ENDPOINT_ADDR structure, which is used with **mspCreateEndpoint**.

```
typedef struct tag_MUX_ENDPOINT_ADDR
{
    // Address attributes
    DWORD    nTimeslot;    // Timeslot address
} MUX_ENDPOINT_ADDR;
```

For more information, see *Creating the MUX endpoint* on page 29 and the *MSPP Service Developer's Reference Manual*.

RTP_PAYLOAD_MAP

This structure associates an expected payload ID for incoming RTP packets with an endpoint type. Use RTP_PAYLOAD_MAP with the MSP_CMD_RTPFDX_MAP command, which can be sent to video RTP endpoints.

```
typedef struct tag_RTP_PAYLOAD_MAP
{
    DWORD    vocoder;
    DWORD    payload_id;
} RTP_PAYLOAD_MAP;
```

For video endpoints, use the following values in the vocoder field:

Endpoint type	Value
MPEG-4	100 (MSP_CONST_VOCODER_MPEG4_VIDEO)
H.263/RFC 2190	34 (MSP_CONST_VOCODER_H263_2190_VIDEO)
H.263/RFC 2429	101 (MSP_CONST_VOCODER_H263_2429_VIDEO)
H.264	104 (MSP_CONST_VOCODER_H264_VIDEO)

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

RTPRTCP_ENDPOINT_ADDR

This structure defines the RTP IP address and port numbers for an RTP IPv4 endpoint source and destination.

Use RTPRTCP_ENDPOINT_ADDR as a parameter in the MSP_ENDPOINT_ADDR structure, which is used with **mspCreateEndpoint**.

```
typedef struct tag_RTPRTCP_ENDPOINT_ADDR
{

    // RTP parameters
    char    DestIpAddress[20];    // destination IP address in dot notation
    WORD    nDestPort;            // destination port number
    char    SrcIpAddress[20];     // source IP address in dot notation
    WORD    nSrcPort;             // source port number

} RTPRTCP_ENDPOINT_ADDR;
```

For more information, see *Creating an endpoint* on page 31 and the *MSPP Service Developer's Reference Manual*.

RTPRTCP_ENDPOINT_PARMS

This structure sets configuration parameters for an RTP IPv4 endpoint. Use this structure as a parameter for the MSP_ENDPOINT_PARAMETER 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;
```

When using RTCP for audio/video synchronization purposes, the startRtcp parameter is used as a bit field containing several definitions. Use the following macros 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 these bits...	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 RCTP 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 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 behind the audio data stream. Set this bit to 1, if you know that the video leads audio in the skew between transmitted video and transmitted audio for a synchronization stream pair. 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 value to the offset value from the other half of the stream pair.
RTCP_VIDEO_SKEW	4 - 15	Sets a value between 0 and 4095 ms of video skew that is 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, see *Using RTCP for audio/video synchronization* on page 58, *Creating an endpoint* on page 31, and the *MSPP Service Developer's Reference Manual.*

RTPRTCP_V6_ENDPOINT_ADDR

This structure defines the RTP IP address and port numbers for an RTP IPv6 endpoint source and destination.

Use RTPRTCP_V6_ENDPOINT_ADDR as a parameter in the MSP_ENDPOINT_ADDR structure, which is used with **mspCreateEndpoint**.

```
typedef struct tag_RTPRTCP_V6_ENDPOINT_ADDR
{
// RTPv6 parameters
char       DestIpv6Address[46];   // destination IPv6 address in ASCII notation
WORD       nDestPort;             // destination port number
WORD       ifc_idx;               // network interface index 0-2 (0 - "auto")
char       SrcIpv6Address[46];    // source IPv6 address in ASCII notation
WORD       nSrcPort;              // source port number
} RTPRTCP_V6_ENDPOINT_ADDR
```

For more information, see *Creating an endpoint* on page 31 and the *MSPP Service Developer's Reference Manual*.

RTPRTCP_V6_ENDPOINT_PARMS

This structure sets configuration parameters for an RTP IPv6 endpoint. Use this structure as a parameter for the MSP_ENDPOINT_PARAMETER 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 multiple definitions.

For more information, see *Creating an endpoint* on page 31, *RTPRTCP_ENDPOINT_PARMS* on page 97, and the *MSPP Service Developer's Reference Manual.*

9 H.324M function summary

Setup functions

Use the following functions to set up the H.324M Middleware and set up calls:

Function	Description
h324Initialize	Sets up the H.324M Middleware for use.
h324SetAudioTxPayloadId	Sets the audio RTP payload ID in the transmit direction.
h324SetupCall	Informs the H.324M Middleware that it can begin H.245 negotiations when it is ready.
h324Start	Creates an H.245 stack for the specified MUX endpoint and sets the initial values that are needed to begin an exchange with the client.

Terminal capabilities functions

Use the following functions to set up and monitor terminal capabilities in the H.324M Middleware:

Function	Description
h324GetTermCaps	Queries the H.324M Middleware for local or remote terminal capabilities.
h324SetTermCaps	Sets the local terminal capabilities for the terminal and initiates the transfer of the terminal capabilities set.

Call control and message functions

Use the following functions to monitor calls and send messages to the remote terminal:

Function	Description
h324_h223SkewIndication	Sends an H.223 skew indication to the terminal.
h324FormatEvent	Formats an H.324M event into a string for print diagnostics.
h324ModifyOutVideoChannelParam	Closes an existing video logical channel and then re-opens the channel with user-supplied logical channel parameters.
h324PassthruDTMFMode	Sets the DTMF mode and payload IF for RFC 2833 DTMF events sent on a pass-through channel.
h324PassthruPlayRFC2833	Plays an application-provided RFC 2833 DTMF event onto the IP network in RTP encapsulation.
h324RoundTripDelay	Sends a round trip delay message to the remote terminal.
h324SubmitEvent	Submits Natural Access events from the event queue to the H.324M Middleware.
h324UserIndication	Sends a user indication (UII H.245) message to the terminal.
h324VendorIDIndication	Sends a vendor ID indication to the remote terminal.

Function	Description
h324VideoFastUpdate	Sends a video fast update message to the remote terminal.
h324VideoTemporalSpatialTradeoff	Sends a VideoTemporalSpatialTradeoff message to the remote terminal.

Error handling functions

Use the following functions to manage errors in the H.324M Middleware:

Function	Description
h324LineErrorReporting	Turns on error reporting statistics in the MUX endpoint.
h324SetTrace	Defines the level of tracing for the H.324M Middleware.

Shut down functions

Use the following functions to stop a call or session in the H.324M Middleware:

Function	Description
h324CloseChannel	Closes an existing media channel.
h324Delete	Releases any objects and memory associated with an instance of the H.324M Middleware.
h324EndSession	Directs the H.324M Middleware to terminate the current H.324 session at the end of the call.
h324Stop	Stops the H.324M Middleware for the specified MUX endpoint.

10 H.324M function reference

Using the H.324M function reference

This section provides an alphabetical reference to the H.324M functions. A typical function includes:

Prototype	The prototype is followed by a list of the function's arguments. All of the H.324M functions have the DWORD (16-bit unsigned) data type. If a function uses an H.324M 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. For a list of the errors returned by H.324M functions, refer to the *Alphabetical error summary* on page 159.
Events	H.3234M events are returned in the application's event buffer. Additional information such as reason codes and return values, appear in the value field of the event. For detailed information about H.324M events, refer to the *Alphabetical event summary* on page 152.
Example	Example functions are taken from sample application programs shipped with the product. The notation /* ... */ indicates additional code that is not shown.

Note: In this reference, the H324_BOOL data type is equivalent to the Boolean data type.

How H.324M functions work

For each function except **h324Initialize**, the host application passes an ***msphd*** parameter that is the handle to a MUX endpoint. The MSPP service returns this handle to the application when the MUX endpoint is created with **mspCreateEndpoint**. The handle is used to determine the channel with which a function is associated.

The H.324M Middleware is asynchronous. No H.324M Middleware functions block while waiting for a response from the CG board or from the remote 3G-324M video terminal. Information from the CG board or the remote 3G-324M video terminal is sent to the application using its Natural Access event queue.

h324_h223SkewIndication

Sends an H.223 skew indication to the terminal.

Prototype

DWORD **h324_h223SkewIndication** (MSPHD ***msphd***, unsigned int ***skewInMs***, channelSkewType ***skewType***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>msphd</i></b></td><td>MSPP handle associated with the MUX endpoint.</td></tr>
<tr><td><b><i>skewInMs</i></b></td><td>Milliseconds of audio or video skew to indicate to the terminal.</td></tr>
<tr><td><b><i>skewType</i></b></td><td>Whether video should be skewed to align with late audio or audio should be skewed to align with late video:
<pre>
typedef struct tag_H324_223_SKEW_INDICATION
{
    channelSkewType   skewType;
    unsigned int   skewInMs;
    unsigned int   logicalChannelNumber1;
    unsigned int   logicalChannelNumber2;
} H324_H223_SKEW_INDICATION;
</pre>
For information about the fields in this structure, see <i>H324_H223_SKEW_INDICATION</i> on page 139.</td></tr>
</table>

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Example

```
rc = h324_h223SkewIndication(GwConfig[0].MuxEp.hd, skewInMs,
                             (selection == 'a') ? audioLate : videoLate);
if (rc == SUCCESS)
    printf("The SkewIndication message was sent successfully.\n");
else
    printf("Failed to send the SkewIndication message...\n");
```

h324CloseChannel

Closes an existing media channel.

Prototype

DWORD **h324CloseChannel** (MSPHD ***msphd***, DWORD ***channel***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
channel	Specifies which of the following channels to close: H324RSN_AUDIO_IN - Incoming unidirectional audio channel H324RSN_AUDIO_OUT - Outgoing unidirectional audio channel H324RSN_VIDEO - Bidirectional video channel H324RSN_VIDEO_IN – Incoming unidirectional video channel H324RSN_VIDEO_OUT – Outgoing unidirectional video channel

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_CHANNEL_CLOSED	Either the host application or the remote terminal successfully closed a channel. The size field defines which channel was closed.
H324EVN_CHANNEL_CLOSE_FAILED	Remote terminal rejected the request to close the specified channel.

Example

```
ret = h324CloseChannel( GwConfig[nGw].MuxEp.hd, H324RSN_VIDEO );
if( ret != SUCCESS )
{
    printf("Failed to close the video logical channel...\n");
    return ret;
}
ret = WaitForSpecificEvent( nGw, GwConfig[nGw].hCtaQueueHd,
                            H324EVN_CHANNEL_CLOSED, &event );
```

h324Delete

Releases any objects and memory associated with an instance of the H.324M Middleware. **h324Delete** must be called after H324EVN_STOP_DONE is received.

Prototype

DWORD **h324Delete** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Example

```
case H324EVN_STOP_DONE:
    printf("H324EVN_STOP_DONE received - Delete the H324M instance now\n");
    ret = h324Delete(pCfg->MuxEp.hd);
    if(ret != SUCCESS)
        {
            printf("h324Delete returned 0x%08x\n",ret);
            return ret;
        }
```

h324EndSession

Directs the H.324M Middleware to terminate the current H.324 session at the end of the call.

Prototype

DWORD **h324EndSession** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_END_SESSION_DONE	End session message was delivered to the remote terminal.

Example

```
ret = h324EndSession( pCfg->MuxEp.hd );
if(ret != SUCCESS)
{
    printf("h324EndSession returned 0x%08x\n",ret);
    return ret;
}
ret = WaitForSpecificEvent( nGw, GwConfig[nGw].hCtaQueueHd,
                            H324EVN_END_SESSION_DONE, &event );
if ( ret != 0 )
{
    printf(“h324EndSession failed to get remote ACK\n");
    return ret;
}
```

h324FormatEvent

Formats an H.324M event into a string for print diagnostics.

Prototype

DWORD **h324FormatEvent** (char ***lineprefix***, CTA_EVENT ***event***, char ***buffer***, unsigned ***size***)

Argument	Description
lineprefix	Pointer to a character string that is placed at the beginning of every new line in the formatted event string.

event	Pointer to the Natural Access CTA_EVENT structure, which is an event structure to be formatted:
	`typedef struct`
	`{`
	`   DWORD   id;            /* Event code and source service ID       */`
	`   CTAHD   ctahd;         /* Natural Access context handle          */`
	`   DWORD   timestamp;     /* Timestamp                              */`
	`   DWORD   userid;        /* Userid (defined by ctaCreateContext)   */`
	`   DWORD   size;          /* Size of buffer if buffer != NULL       */`
	`   void    *buffer;       /* Buffer pointer                         */`
	`   DWORD   value;         /* Event status or event-specific data    */`
	`   DWORD   objHD;         /* Service object handle                  */`
	`} CTA_EVENT;`

buffer	Pointer to the buffer to receive the event string.
size	Size of the buffer in bytes.

Return values

Return value	Description
SUCCESS	

Events

None.

Details

h324FormatEvent creates a detailed textual description of the H.324M event.

See also

h324SubmitEvent

Example

```
void DemoShowEvent( CTA_EVENT *event )
{
    char format_buffer[CTA_MAX_FORMAT_SIZE];
    char *prefix = "\t\t\t";  /* default demo indent */
    format_buffer[0] = '\0';
    h324FormatEvent( prefix, event, format_buffer, CTA_MAX_FORMAT_SIZE );
    printf( "%s", format_buffer );
}
```

h324GetTermCaps

Queries the H.324M Middleware for local or remote terminal capabilities.

Prototype

DWORD **h324GetTermCaps** (MSPHD ***msphd***, WORD ***location***, H324_TERM_CAPS ***termCaps**)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint..
location	Indicates whether the function applies to the local terminal capabilities set or the remote terminal capabilities set. Valid values are: H324_LOCAL_TERMINAL - Local terminal capabilities set. H324_REMOTE_TERMINAL - Remote terminal capabilities set.

termCaps	Pointer to the H324_TERM_CAPS structure, which communicates the MUX and media capabilities of the remote terminal and the host application:
	`typedef struct tag_H324_TERM_CAPS`
	`{`
	`   DWORD                    size;`
	`   MULTIPLEX_CAPABILITY     muxCap;`
	`   DWORD                    wCapCount;  // Number of caps`
	`   H324_MEDIA_CAPABILITY    capTable[16];`
	`} H324_TERM_CAPS;`
	For information about the fields in this structure, see *H324_TERM_CAPS* on page 144.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Details

For the outbound terminal capabilities set, use **h324GetTermCaps** to retrieve the default set of terminal capabilities structures based on either the:

- List of capabilities defined in the start parameters structure.
- Default list of AMR and MPEG-4 (if no start parameters were provided).

You then edit specific parameters in these structures as required, and pass them back to the H.324M Middleware using **h324SetTermCaps**.

For the inbound terminal capabilities set, you can retrieve a copy of the set at any time after receiving the H324EVN_REMOTE_CAPS. If you issue this request before receiving the H324EVN_REMOTE_CAPS, the function returns H324ERR_INTERNAL_ERROR.

If no terminal capabilities have been set by the application, the default terminal capabilities are returned.

Example

```
ret = h324GetTermCaps( pCfg->MuxEp.hd, H324_REMOTE_TERMINAL, &remoteTermCaps);
if ( ret != SUCCESS)
{
    printf("h324GetTermCaps. Return code is %x\n", ret);
    return ret;
}
    printf("Call to Remote h324GetTermCaps returned %d capabilities\n",
           remoteTermCaps.wCapCount );
```

h324Initialize

Sets up the H.324M Middleware for use. This function must be called only once by the application and must not be called by more than one thread.

Prototype

DWORD **h324Initialize** (char ***logFileName***)

Argument	Description
logFileName	Pointer to the H.324 log file name. If NULL, the default file name of *h324.log* is used.

Return values

Return value	Description
SUCCESS	
H324ERR_ALREADY_INITIALIZED	**h324Initialize** was called.
H324ERR_LOG_FILE_OPEN_FAILED	Error opening the log file.
H324ERR_MUTEX_CREATE_FAILED	Failed to create mutex semaphore.

Events

None.

Example

```
ret = h324Initialize(gwParm.sH324LogFile);
if ( ret != SUCCESS)
{
    printf("h324Initialize . Return code is %x\n", ret);
    exit(1);
}
```

h324LineErrorReporting

Turns on error reporting statistics in the H.223 demultiplexer. Use the statistics to determine the quality of the inbound radio link from the H.324 terminal to the demultiplexer.

Prototype

DWORD **h324LineErrorReporting** (MSPHD ***msphd***, DWORD ***command***, WORD ***param1***, WORD ***param2***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>msphd</i></b></td><td>MSPP handle associated with the MUX.</td></tr>
<tr><td><b><i>command</i></b></td><td>Valid values are:
H324_LINE_STAT_CMD_ERROR_EVENT - Sends an error event when the specified error is detected.
H324_LINE_STAT_CMD_PERIODIC - Periodically reports line error counts.
H324_LINE_STAT_CMD_RESET_STAT - Resets all line error counts to zero.</td></tr>
<tr><td><b><i>param1</i></b></td><td><b><i>param1</i></b> value depends upon the value selected for <b><i>command</i></b>:
<table>
<tr><th>If <i>command</i> is...</th><th>The <i>param1</i> value...</th></tr>
<tr><td>H324_LINE_STAT_CMD_ERROR_EVENT</td><td>Enables or disables specific error reporting. Valid values are:
0 - Disable
1 - Enable</td></tr>
<tr><td>H324_LINE_STAT_CMD_PERIODIC</td><td>Enables or disables periodic error reporting. Valid values are:
0 - Disable
1 - Enable</td></tr>
<tr><td>H324_LINE_STAT_CMD_RESET_STAT</td><td>Is not used.</td></tr>
</table></td></tr>
<tr><td><b><i>param2</i></b></td><td><b><i>param2</i></b> value depends upon the value selected for <b><i>command</i></b>:
<table>
<tr><th>If <i>command</i> is...</th><th>The <i>param2</i> value...</th></tr>
<tr><td>H324_LINE_STAT_CMD_ERROR_EVENT</td><td>Is a bit mask that selects which error triggers an error report event. Bit mask values are:
1 - Video CRC error
2 - Audio CRC error
4 - Golay coding error in PDU header</td></tr>
<tr><td>H324_LINE_STAT_CMD_PERIODIC</td><td>Defines the time in seconds between periodic reports.</td></tr>
<tr><td>H324_LINE_STAT_CMD_RESET_STAT</td><td>Is not used.</td></tr>
</table></td></tr>
</table>

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description

MSPEVN_DEMUX_CRC_ERR_REPORTS	Demux detected a line error, and is sending event data in a structure of type MSP_ENDPOINT_DEMUX_CRC_ERROR_REPORTS:
	`typedef struct`
	`tag_msp_ENDPOINT_DEMUX_CRC_ERROR_REPORTS`
	`{`
	`    DWORD FilterId;`
	`    WORD  EvnBase;`
	`    WORD  EvnId;`
	`    DWORD error_check_type;`
	`              // 1 = video CRC check`
	`              // 2 = audio CRC check`
	`              // 4 = header Golay coding check`
	`    DWORD total_num_errors;`
	`} msp_ENDPOINT_DEMUX_CRC_ERROR_REPORTS;`

MSPEVN_DEMUX_GET_PERIODIC_STATS	Demux sent its periodic error statistics. The buffer contains a structure of type MSP_ENDPOINT_DEMUX_PERIODIC_STATS:
	`typedef struct`
	`tag_msp_ENDPOINT_DEMUX_PERIODIC_STATS`
	`{`
	`    DWORD   FilterId;`
	`    WORD    EvnBase;`
	`    WORD    EvnId;`
	`    DWORD   num_videoCRCerrors;`
	`    DWORD   num_videoCRCsuccesses;`
	`    DWORD   num_audioCRCerrors;`
	`    DWORD   num_audioCRCsuccesses;`
	`    DWORD   num_headerGolayerrors;`
	`    DWORD   num_headerGolaysuccesses;`
	`    DWORD   reserved1;`
	`    DWORD   reserved2;`
	`} msp_ENDPOINT_DEMUX_PERIODIC_STATS;`

Example

```
printf("\n\tEnable/Disable Periodic Statistics (0=Disable, 1=Enable) > ");
fflush(stdin);
scanf("%hd", &value1);
printf("\n\tSet Periodic Statistics Interval (1-10 seconds X 10) > ");
fflush(stdin);
scanf("%hd", &value2);
h324LineErrorReporting( GwConfig[0].MuxEp.hd,
                        H324_LINE_STAT_CMD_PERIODIC,
                        value1,
                        value2);


...


// received an asynchronous event: CTA_EVENT *pevent
switch (pevent->value)
{
  case (MSPEVN_DEMUX_PERIODIC_STATS):
     if((GwConfig[nGw].MuxEp.hd == pevent->objHd) && (pevent->buffer != NULL)
       && (pevent->size > 0))
       ShowDemuxPeriodicStats(nGw, (msp_ENDPOINT_DEMUX_PERIODIC_STATS *)pevent->buffer);
       break;

  case (MSPEVN_DEMUX_CRC_ERR_REPORT):
     if(GwConfig[nGw].MuxEp.hd == pevent->objHd)
     {
      msp_ENDPOINT_DEMUX_CRC_ERROR_REPORTS *err
= (msp_ENDPOINT_DEMUX_CRC_ERROR_REPORTS*)pevent->buffer;
       switch( NMS2H_DWORD(err->error_check_type
        {
          case 1:  // Video CRC Error
            printf("VIDEO CRC ERROR Received at Demux,
                   Error Count = %d\n",
                   NMS2H_DWORD(err->total_num_errors)  );
            break;
          case 2:  // Audio CRC Error
            printf("AUDIO CRC ERROR Received at Demux, Error Count = %d\n",
                   NMS2H_DWORD(err->total_num_errors) );
            break;
          case 4:  // Golay
            printf("HEADER GOLAY CODING ERROR Received at Demux,
                   Error Count = %d\n",
                   NMS2H_DWORD(err->total_num_errors) );
                   break;
        }
     }
break;
```

h324ModifyOutVideoChannelParam

Closes an existing video logical channel and then re-opens the channel with user-supplied logical channel parameters.

Prototype

DWORD **h324ModifyOutVideoChannelParam** (MSPHD ***msphd***, MEDIA_UNION* ***pMediaParam***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
pMediaParam	Pointer to MEDIA_UNION structure that specifies the new parameters with which the video logical channel is re-opened. For information about the fields in this structure, see *Fields in the MEDIA_UNION structure* on page 141.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	Provided ***msphd*** does not map to a current video logical channel. Either **h324Start** was not called, **h324Stop** was called, or the wrong handle was passed by the application.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Event

Event	Description
H324EVN_CHANNEL_CLOSED	A state transition information event indicating that the video logical channel was closed. **Note:** Unlike the other events for **h324ModifyOutVideoChannelParam**, this event does not signal the completion of the modification procedure for the channel parameters.
H324EVN_CHANNEL_CLOSE_FAILED	The bi-directional channel could not be closed. It is still in active state.
H324EVN_LCD	Video channel was properly re-established.
H324EVN_MEDIA_SETUP_FAILED	An issue occurred when attempting to close or re-establish the video channel. The video channel is in inactive state.

Example

```
BYTE h264_dci[] =
{
    0x00, 0x00, 0x00, 0x01, 0x27, 0x42, 0xe0, 0x0a,
    0x95, 0xa0, 0xb1, 0x3a, 0x01, 0xfd, 0x40, 0x00,
    0x00, 0x00, 0x01, 0x28, 0xce, 0x06, 0x6a
};
int   h264_dci_len = 23;
BYTE* l_pHdrCfg    = h264_dci;
int   l_nHdrCfgLen = h264_dci_len;
H324_MEDIA_CAPABILITY* pH264Cap;

pH264Cap = &GwConfig[0].localTermCaps.capTable[1];
pH264Cap->u.h264.capability.bit_mask               |= decoder_config_info_present;
pH264Cap->u.h264.capability.decoder_config_info_len = l_nHdrCfgLen;
pH264Cap->u.h264.capability.decoder_config_info     = new unsigned char[l_nHdrCfgLen];
memcpy( pH264Cap->u.h264.capability.decoder_config_info, l_pHdrCfg, l_nHdrCfgLen );

if (h324ModifyOutVideoChannelParam( GwConfig[0].MuxEp.hd,
    &GwConfig[0].localTermCaps.capTable[1].u ) != SUCCESS)
{
    printf("ERROR: h324ModifyOutVideoChannelParam () failed.\n");
}
```

h324PassthruDTMFMode

Sets the DTMF mode and payload ID for RFC 2833 DTMF events sent on a pass-through channel.

Prototype

DWORD **h324PassthruDTMFMode** (MSPHD ***msphd***, WORD ***control***, WORD ***payload***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
control	Sets the DTMF mode for the associated pass-through channel. Valid values are: 0 - Does not shift the timestamp of associated DTMF RTP packets. DTMF_SHIFT_ENABLED - Shifts the timestamp of associated DTMF RTP packets back to better align with the actual start of the DTMF detection. This allows for more synchronized decoder playout of RFC 2833 DTMF digits with respect to the actual DTMF occurrence.
payload	Sets the payload ID for the RFC 2833-compliant DTMF packets sent through the associated pass-through channel. The range is 96 - 127. The default value is 96.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** does not map to a current audio pass-through channel. Either **h324Start** was not called, **h324Stop** was called, or the wrong handle was passed by the application.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Event

Event	Description
H324EVN_PASSTHRU_DTMF_MODE_DONE	DTMF mode and payload ID were configured.

See also

h324PassthruPlayRFC2833

Example

```
{
    WORD control, payloadID;
    payloadID = 97;
    control = DTMF_SHIFT_ENABLED;
    if (h324PassthruDTMFMode(GwConfig[0].MuxEp.hd, control, payloadID) != SUCCESS)
        printf("ERROR: h324PassthruDTMFMode() failed.\n");
}
```

h324PassthruPlayRFC2833

Plays an application-provided RFC 2833 DTMF event onto the IP network in RTP encapsulation. For more information, refer to *Transferring DTMF digits according to RFC 2833* on page 42.

Prototype

DWORD **h324PassthruPlayRFC2833** (MSPHD ***msphd***, WORD ***event***, WORD ***duration***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
event	RFC 2833 event (0 - 255) to send. Refer to the RFC 2833 specification for information about available options.
duration	Duration of the RFC2833 event, in ms.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** does not map to a current audio pass-through channel. Either **h324Start** was not called, **h324Stop** was called, or the wrong handle was passed by the application.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_PASSTHRU_PLAY_RFC2833_DONE	MSPP service accepted the command to send the specified DTMF digit in RFC 2833 encapsulation, and forwards the command to the runtime software on the CG board.

See also

h324PassthruDTMFMode

Example

```
{
    DWORD result;
    WORD nEventID, nEventDuration;
    nEventID = ‘3’; // The keypad digit ‘3’
    EventDuration = 80; // Digit pressed for 80 ms.
    result = h324PassthruPlayRFC2833(GwConfig[0].MuxEp.hd, nEventID, EventDuration);
    if (result != SUCCESS)
        printf("ERROR: h324PassthruPlayRFC2833() failed.\n");
}
```

h324RoundTripDelay

Sends a round trip delay message to the remote terminal.

Prototype

DWORD **h324RoundTripDelay** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Neither **h324Start** nor **h324Delete** were called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_ROUND_TRIP_DELAY	Remote terminal responded to the round trip delay message.
H324EVN_ROUND_TRIP_TIMEOUT	Remote terminal did not respond.

Example

```
ret = h324RoundTripDelay( GwConfig[0].MuxEp.hd );
if ( ret != SUCCESS)
{
    printf("h324RoundTripDelay. Return code is %x\n", ret);
    return ret;
}
```

h324SetAudioTxPayloadID

Sets the audio RTP payload ID in the transmit direction.

Prototype

DWORD **h324SetAudioTxPayloadID** (MSPHD ***msphd***, WORD ***payloadID***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
payloadID	Audio RTP payload ID.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	Provided ***msphd*** does not map to a current audio pass-through channel. Either **h324Start** was not called, **h324Stop** was called, or the wrong handle was passed by the application.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Example

```
{
DWORD result;
WORD  payloadID;
payloadID = 100; // Set AudioRTP payload ID to 100
result = h324SetAudioTxPayloadID(GwConfig[0].MuxEp.hd, payloadID);
if (result != SUCCESS)
printf("ERROR: h324SetAudioTxPayloadID () failed.\n");
}
```

h324SetTermCaps

Sets the local terminal capabilities for the terminal, and initiates the transfer of the terminal capabilities set if **h324SetupCall** has already been called. These values are used in the terminal capabilities exchange and in the open logical channel messages.

Prototype

DWORD **h324SetTermCaps** (MSPHD ***msphd***, H324_TERM_CAPS ****termCaps***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>msphd</i></b></td><td>MSPP handle associated with the MUX endpoint.</td></tr>
<tr><td><b><i>termCaps</i></b></td><td>Pointer to H324_TERMCAPS structure that communicates the MUX and media capabilities of the both the remote terminal and the host application:
<pre>
typedef struct tag_H324_TERM_CAPS
{
   DWORD   size;
   MULTIPLEX_CAPABILITY   muxCap;
   DWORD   wCapCount; // Number of caps
   H324_MEDIA_CAPABILITY   capTable[16];
} H324_TERM_CAPS;
</pre>
For information about the fields in this structure, see <i>H324_TERM_CAPS</i> on page 144.</td></tr>
</table>

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Details

h324SetTermCaps allows you to modify capability parameters defined in h324Start.

If ***termCaps*** = NULL, the default terminal capabilities are used.

The default terminal capabilities set supports video with AL3 and AL2.

Example

```
ret = h324GetTermCaps( pCfg->MuxEp.hd, H324_LOCAL_TERMINAL, &pCfg->localTermCaps);
if ( ret != SUCCESS)
{
  printf("h324GetTermCaps. Return code is %x\n", ret);
  return ret;
}
printf("Call to Local h324GetTermCaps returned %d capabilities\n",
       pCfg->localTermCaps.wCapCount );

ret = h324SetTermCaps( pCfg->MuxEp.hd, &pCfg->localTermCaps);
{
  printf("h324GetTermCaps. Return code is %x\n", ret);
  return ret;
}
```

h324SetTrace

Defines the level of tracing for the H.324M Middleware. Applications can invoke **h324SetTrace** any time before or after invoking **h324Initialize**.

Prototype

DWORD **h324SetTrace** (unsigned int ***LogToConsoleMask***, unsigned int ***LogToFileMask***)

Argument	Description
LogToConsoleMask	Tracing mask for the console.
LogToFileMask	Tracing mask for the log file.

The following bits can be turned on or off for both ***LogToConsoleMask*** and ***LogToFileMask***:

Bit	Value	What gets logged
T_H245	0x00001	H.245 negotiations.
T_H245INFO	0x00002	Contents of some H.245 messages in binary form.
T_H245ERR	0x00004	H.245 stack errors.
T_H245ECHO	0x00008	Information about the H.324M Middleware functions called by the application.
T_EVENT	0x00010	Events processed by **h324SubmitEvent**.
T_APPEVENT	0x00020	Events generated by the H.324M Middleware.
T_APIECHO	0x00040	A message every time certain H.324M Middleware functions are called.
T_APIERR	0x00080	Additional information in cases of H.324M Middleware failures.
T_APIINFO	0x00100	Information about events consumed by the H.324M Middleware.
T_MUTEX	0x00200	A message every time the internal H.324M mutex is locked or unlocked.
T_MUXIFINFO	0x00400	Additional information related to the communication with the MUX or DEMUX.
T_MUXIFECHO	0x00800	MUX or DEMUX interface functions.
T_MUXIFERR	0x01000	Errors while communicating with the MUX or DEMUX.
T_H245MSGSUM	0x02000	NSRP related information.
T_H245MSGDET	0x04000	Detailed contents of H.245 messages received from the DEMUX in the binary form.
T_H245CALLB	0x08000	H.245 stack callback functions.

Bit	Value	What gets logged
T_OLC	0x10000	Audio or video OLC state machine.
T_STACKINFO	0x20000	Reserved for future use.
T_STACKWARN	0x40000	Reserved for future use.
T_STACKERROR	0x80000	Reserved for future use.
T_ALL	0xFFFFF	Full tracing.
T_ALLERR	T_H245ERR + T_APIERR + T_MUXIFERR + T_STACKERROR + T_STACKWARN	Errors only.
T_NONE	0x00000	No tracing.

Return values

Return value	Description
SUCCESS	

Events

None.

Example

To print errors only to the log file, use **h324SetTrace** in the following way:

```
H324SetTrace( 0, T_ALLERR );
```

h324SetupCall

Informs the H.324M Middleware that it can begin H.245 negotiations when it is ready. This usually occurs when PMSYNC stops, indicating that the local and remote multiplexers are synchronized. For more information, see *Setting up a 3G-324M session* on page 29.

Prototype

DWORD **h324SetupCall** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_MEDIA_SETUP_DONE	H.245 negotiations completed

Example

```
case H324EVN_START_DONE:
    printf("Calling h324SetupCall\n");
    ret = h324SetupCall(pCfg->MuxEp.hd);
    if(ret != SUCCESS)
    {
        printf("Error:h324SetupCall return 0x%08x\n",ret);
        return ret;
    }
```

h324Start

Creates an H.245 stack for the specified MUX endpoint and sets the initial values needed to begin an exchange with the client (such as resetting PMSYNC and the video sequence number).

Prototype

DWORD **h324Start** (CTAHD ***ctahd***, MSPHD ***msphd***, H324_START_PARAMS ****params***)

Argument	Description
ctahd	CTA handle for the application event queue.
msphd	MSPP handle associated with the MUX endpoint.

params	Pointer to the H324_START_PARAMS structure that holds initial parameters for the H.324 session:
	`typedef struct tag_H324_START_PARAMS`
	`{`
	`   DWORD         size;`
	`   int           terminalType;`
	`   int           iCapCount;`
	`   int           iCap[16];`
	`   H324_BOOL     bAutoChannelSetup;`
	`} H324_START_PARAMS;`
	For information about the fields in this structure, see *H324_START_PARAMS* on page 143.

Return values

Return value	Description
SUCCESS	
H324ERR_INITIALIZE_STACK_FAILED	Unable to initialize the H.245 stack.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_FAST_CALL_STEUP_DONE	MONA MPC fast call setup is complete, and the H.324M Middleware was initialized for the specified MUX.
H324EVN_START_DONE	H.324M Middleware was initialized for the specified MUX, and provides one of the following status codes: CTA_REASON_FINISHED – Successful call to **h324Start**. The H.324M Middleware was initialized for the specified MUX endpoint. CTA_REASON_TIMEOUT – H.223 level synchronization problem with the remote terminal. The initialization of the MUX endpoint failed.

Details

If no parameters are provided, a set of defaults is used. The parameters contain an array of the following capability types (iCap):

Value	Description
0	H324_MPEG4_VIDEO
1	H324_H263_VIDEO
2	H324_H261_VIDEO (Not supported)
3	H324_G711_AUDIO (Not supported)
4	H324_G723_AUDIO
5	H324_AMR_AUDIO
6	H324_H264_VIDEO

Use the array to specify a terminal capabilities set that you can then modify.

When the H.324M Middleware is initialized for the specified MUX, the application receives H324EVN_FAST_CALL_STEUP_DONE instead of H324EVN_START_DONE, if it used MONA MPC for fast call setup.

See also

h324Stop

Example

```
Example to declare a single Audio codec capability and single Video codec capability
H324_START_PARAMS params;
params.size         = sizeof(params);
params.iCapCount    = 2;
params.terminalType = 201;
params.iCap[0] = H324_AMR_AUDIO;
params.iCap[1] = H324_MPEG4_VIDEO;
params.bAutoChannelSetup = TRUE;

ret = h324Start(pCtx->ctahd, pCfg->MuxEp.hd, &params);
if(ret != SUCCESS)
{
    printf("Error: h324Start returned 0x%08x\n",ret);
    return ret;
}
```

h324Stop

Stops the H.324M Middleware for the specified MUX endpoint. Call this function at the end of a 3G-324M call.

Prototype

DWORD **h324Stop** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Neither **h324Start** nor **h324Delete** were called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

Event	Description
H324EVN_STOP_DONE	H.324M Middleware was stopped, and provides a status code for success or failure. The application calls **h324Delete** when this event is received.

Example

```
ret = h324Stop(GwConfig[i].MuxEp.hd);
if (ret != SUCCESS)
    {
        printf("h324Stop returned 0x%x.\n", ret);
        return ret;
    }
else
    {
    ret = WaitForSpecificEvent ( i, GwConfig[i].hCtaQueueHd, H324EVN_STOP_DONE, &event);
    if (ret != 0)
        {
            printf("H324EVN_STOP_DONE event failure, ret= %x, waitEvent.value=%x\n",
                   ret, event.value);
            return ret;
        }
      }
```

h324SubmitEvent

Submits a Natural Access event from the event queue to the H.324M Middleware.

Prototype

DWORD **h324SubmitEvent** (CTA_EVENT ***event**, H324_BOOL ***consumed**)

Argument	Description
event	Pointer to an event received from **ctaWaitEvent**.
consumed	Pointer to a returned value indicating whether the H.324M Middleware consumed the event.

Return values

Return value	Description
SUCCESS	
H324ERR_COMMAND_RESPONSE_ERROR	A command done event was received from an unexpected command.
H324ERR_INCOMING_MSG_ERROR	Error processing an incoming H.245 message.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_PMSYNC_ERROR	Error after multiplexer synchronizes with remote multiplexer.

Events

None.

Details

h324SubmitEvent can either consume or not consume each event submitted to it. If the event is consumed, the application continues to wait for additional Natural Access events. If the event is not consumed, it is an event for the application, which should process it normally.

The application does not need to release MSPP buffers associated with events that are consumed by **h324SubmitEvent**. These buffers are released by the H.324M Middleware.

See also

h324FormatEvent

Example

```
CTA_EVENT event;
H324_BOOL consumed;
while(1)
    {
    ctaWaitEvent(ctaqueuehd, &event, CTA_WAIT_FOREVER);
    ret = h324SubmitEvent(&event, &consumed);
    if(ret != SUCCESS)
        {
        /* Error processing */
        }
    else if(consumed)
    continue;

    /* Normal application event processing goes here */
    }
```

h324UserIndication

Sends a user indication (UII H.245) message to the terminal.

Prototype

DWORD **h324UserIndication** (MSPHD ***msphd***, H324_USER_INPUT_INDICATION ****msg***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>msphd</i></b></td><td>MSPP handle associated with the MUX endpoint.</td></tr>
<tr><td><b><i>msg</i></b></td><td>Pointer to the H324_USER_INPUT_INDICATION structure that defines the user input indication message to send:
<pre>
typedef struct tag_H324_USER_INPUT_INDICATION
{
   char   data[H324_MAX_USER_INDICATION_SIZE];
   DWORD   length;
   char   szObjectId[H324_MAX_UII_OBJECT_ID_SIZE];
   DWORD   msgType;
} H324_USER_INPUT_INDICATION;
</pre>
For information about the fields in this structure, see <i>H324_USER_INPUT_INDICATION</i> on page 148.</td></tr>
</table>

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Neither **h324Start** nor **h324Delete** were called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Details

The user input indication message type can be defined by setting field values as follows:

For this type of message...	Do the following...
Alphanumeric	Set the msgType field to H324_USER_INPUT_ALPHANUMERIC. Set the szObjectID field to 0 (zero).
Non-standard	Set the msgType field to H324_USER_INPUT_NONSTANDARD. Copy a zero terminated string into szObjectID to set the objectidentifier. This string is a dot-separated number sequence, such as: `1.2.123.2345.5.73`
Signal	Set the msgType field to H324_USER_INPUT_SIGNAL. Set the szObjectID field to 0 (zero).

The size for H324_MAX_USER_INDICATION_SIZE = 512. The size for H324_MAX_UII_OBJECT_ID_SIZE = 128.

Example

```
// sends “1234” alphanumeric string to the remote terminal
l_UII.length = 4;
l_UII.msgType = H324_USER_INPUT_ALPHANUMERIC;
l_UII.szObjectId[0] = 0;
strcpy( l_UII.data, "1234");
printf("UII length: %d\n", l_UII.length );
ret = h324UserIndication( GwConfig[Port].MuxEp.hd, &l_UII );
if( ret != SUCCESS )
    printf("Failed to send the alphanumeric UII message...\n");
```

h324VendorIDIndication

Sends a vendor ID indication to the remote terminal.

Prototype

DWORD **h324VendorIDIndication** (MSPHD ***msphd***,
H324_VENDORID_INDICATION ***pVendorInd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

pVendorInd	Pointer to the H_324_VENDORID_INDICATION structure that describes the VendorID indication to transmit:
	`typedef struct tag_H324_VENDORID_INDICATION`
	`{`
	`  unsigned short    vendorIDLen;`
	`  unsigned short    productNumberLen;`
	`  unsigned short    versionNumberLen;`
	`  unsigned char     isNonStandard;`
	`  unsigned char     bytes[sizeof(char)];`
	`  // "vendorID" field resides at bytes[0] through bytes[vendorIDLen-1].`
	`  // OPTIONAL "productNumber" resides at`
	`  // bytes[vendorIDLen] through`
	`  // bytes[vendorIDLen+productNumberLen-1].`
	`  // OPTIONAL "versionNumber" resides at`
	`  // bytes[vendorIDLen+productNumberLen]`
	`  // through bytes[vendorIDLen+productNumberLen+versionNumberLen-1].`
	`} H324_VENDORID_INDICATION;`
	For information about the fields in this structure, see *H324_VENDORID_INDICATION* on page 149.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Neither **h324Start** nor **h324Delete** were called.
H324ERR_INTERNAL_ERROR	Internal error in the 3G-324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Example

```
#include <h324def.h>

unsigned char vendor_id[] = { 1, 28, 111, 97, 111, 84, 43, 86, 49 };
char prod_num[] =  "N5000";
char vers_num[] =  "1.23.5";
enum vendorIdType { t_ObjectId, t_h221NonStandard };
int sendVendorIDIndication(unsigned char * vendor_p, unsigned vendor_len,
unsigned char * prodNum_p, unsigned prodNum_len,
unsigned char * verNum_p, unsigned verNum_len,
vendorIdType type, MSPHD msphd)
{
    unsigned result, payloadSz = vendor_len + prodNum_len + verNum_len;
    H324_VENDORID_INDICATION * pVIBuf;

    if (!(pVIBuf = (H324_VENDORID_INDICATION *)malloc(
        sizeof(H324_VENDORID_INDICATION)+payloadSz)))
        return -1;
        memset(pVIBuf, 0, sizeof(H324_VENDORID_INDICATION) + payloadSz);

    // VendorID is either an ObjectID or H.221 NonStandard identifer.
    pVIBuf->isNonStandard = (type == t_h221NonStandard);

    // VendorID field lengths are required.
    pVIBuf->vendorIDLen = vendor_len;
    pVIBuf->productNumberLen = prodNum_len;  // MBZ unless productNumber populated
    pVIBuf->versionNumberLen = verNum_len;   // MBZ unless versionNumber populated

    // Copy required vendor ID into VendorID indication buffer.
    memcpy(&pVIBuf->bytes[0], vendor_p, vendor_len);

    // Copy optional productNumber and versionNumber fields, if present.
    if (prodNum_len)
        memcpy(&pVIBuf->bytes[vendor_len], prodNum_p, prodNum_len);
    if (verNum_len)
        memcpy(&pVIBuf->bytes[vendor_len + prodNum_len], verNum_p, verNum_len);

    // Send the VendorID indication.
    result = h324VendorIDIndication(msphd, pVIBuf);
    free(pVIBuf);

    if (result != SUCCESS)
        return -2;

    return 0;       // VendorID successfully sent.
}
```

h324VideoFastUpdate

Sends a video fast update message to the remote terminal.

Prototype

DWORD **h324VideoFastUpdate** (MSPHD ***msphd***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Neither **h324Start** nor **h324Delete** were called.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Example

```
ret = h324VideoFastUpdate( GwConfig[nGw].MuxEp.hd );
if(ret != SUCCESS) {
    printf("Error:h324SetupMedia returned 0x%08x\n",ret);
    return ret;
}
```

h324VideoTemporalSpatialTradeoff

Sends a VideoTemporalSpatialTradeoff message to the remote terminal.

Prototype

DWORD **h324VideoTemporalSpatialTradeoff** (MSPHD ***msphd***, unsigned short ***tradeoff***)

Argument	Description
msphd	MSPP handle associated with the MUX endpoint.
tradeoff	Temporal-spatial tradeoff value between 0 and 31 (unsigned integer).

Return values

Return value	Description
SUCCESS	
CTAERR_NOT_FOUND	***msphd*** was not found. Either **h324Start** was not called or **h324Delete** was called.
H324ERR_INTERNAL_ERROR	Internal error in the H.324M Middleware.
H324ERR_MUTEX_LOCK_FAILED	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	**h324Initialize** was not called first.

Events

None.

Details

A tradeoff value of zero indicates a high spatial resolution. A tradeoff value of 31 indicates a high frame rate. ***msphd*** identifies the MUX endpoint and the logical channel number of the associated outbound video channel.

Per H.245, values from 0 to 31 indicate monotonically a higher frame rate, and actual values do not correspond to precise values of spatial resolution or frame rate.

Example

```
ret = h324VideoTemporalSpatialTradeoff(GwConfig[0].MuxEp.hd, tradeoff);
if (ret == SUCCESS)
    printf("The videoTemporalSpatialTradeoff Indication message was sent
            successfully.\n");
else
    printf("Failed to send the videoTemporalSpatialTradeoff Indication message...\n");
```

11 H.324M structure reference

Using the H.324M structure reference

This section provides an alphabetical reference to the structures used by H.324M functions in the 3G-324M Interface. The topics in the structure reference include the structure definition and a table of field descriptions.

The table below lists the H.324M structures by function and event. Only those functions and events that have associated structures are included in the list.

H.324M functions and events	Associated structures
H324EVN_FAST_CALL_SETUP_DONE	H324_FAST_CALL_SETUP_PARAMS
H324EVN_H223_SKEW_INDICATION	H324_H223_SKEW_INDICATION
H324EVN_LCD	H324_LCD
h324Start	H324_START_PARAMS
h324SetTermCaps H324EVN_REMOTE_CAPABILITIES	H324_TERM_CAPS
h324UserIndication H324EVN_USER_INDICATION	H324_USER_INPUT_INDICATION
h324VendorIDIndication H324EVN_VENDOR_INDICATION	H324_VENDORID_INDICATION
H324EVN_VIDEOTEMPORALSPATIALTRADEOFF _INDICATION	H324_VIDEOTEMPORALSPATIALTRADEOFF _INDICATION

Note: In this reference, the H324_BOOL data type is equivalent to the Boolean data type.

H324_FAST_CALL_SETUP_PARAMS

The 3G-324M Interface uses this structure to notify the host application that a call has been set up using MONA (Media Oriented Negotiation Acceleration). This structure is used with inbound H324EVN_FAST_CALL_SETUP_DONE events.

Upon receiving this event, the host application can start transmitting the media. This reduces the delay introduced by waiting for an H324EVN_MEDIA_SETUP_DONE event.

For more information, see *Enabling fast call setup* on page 23.

Definition

```
typedef struct tag_H324_FAST_CALL_SETUP_PARAMS
{
    int           audioCodec;
    int           videoCodec;
    H324_BOOL     remoteMediaBuffering;
} H324_FAST_CALL_SETUP_PARAMS;
```

Fields

Field name	Description
audioCodec	Choice of audio codec for the incoming audio channel. The valid value is 5 - AMR.
videoCodec	Choice of video codec for the incoming video channel. Valid values are: 0 - MPEG4 1 - H.263 6 - H.264
remoteMediaBuffering	Indicates whether the remote terminal can buffer the incoming media send before receiving relevant OLC messages in the context of MONA accelerated H.245 signaling (ACP) call setup. Buffering the incoming media send results in faster call setup. Values are: TRUE: Remote terminal supports media buffering. FALSE: Remote terminal does not support media buffering.

H324_H223_SKEW_INDICATION

The remote terminal uses this structure to pass skew indications to the host application. This structure is used with inbound H324EVN_H223_SKEW_INDICATION events.

A skew indication indicates the relative delay between two channels. When an application receives an H324EVN_H223_SKEW_INDICATION event, it must apply a corresponding delay to the faster channel to re-synchronize playback of the two channels. The 3G-324M Interface does not provide a re-synchronization mechanism.

For more information about skew indications, see *Using RTCP for audio/video synchronization* on page 58.

Definition

```
typedef struct tag_H324_223_SKEW_INDICATION
{
   channelSkewType    skewType;
   unsigned int       skewInMs;
   unsigned int       logicalChannelNumber1;
   unsigned int       logicalChannelNumber2;
} H324_H223_SKEW_INDICATION;
```

Fields

Field name	Description
skewType	Indicates the type of data that is delayed. Valid values are: audioLate - Audio data is arriving later than the video data. videoLate - Video data is arriving later than the audio data.
skewInMs	Relative delay in ms between the two channels specified in the logicalChannelNumber1 and logicalChannelNumber2 fields. This value can range from 0 to 4095.
logicalChannelNumber1	Channel number of the delayed channel. This value can range from 1 to 65535.
logicalChannelNumber2	Channel number of the faster channel. This value can range from 1 to 65535.

H324_LCD

Conveys information about the local and remote channels between the host application and the remote terminal. The H324_LCD structure is in a set of structures that also includes the MEDIA_UNION structure, which contains media options for specific media types.

H324_LCD is used with inbound H324EVN_LCD events.

The H.324M Middleware uses this structure set to describe the following:

- The final bi-directional channel established by the host application and the remote terminal.
- An incoming unidirectional OLC request accepted by H.324M Middleware.
- An outgoing unidirectional OLC request accepted by the remote terminal.

Definition

```
typedef struct tag_H324_LCD
{
   // Receive Info
   H324_BOOL    bReceive;
   int          rxChannel;
   DWORD        rxChoice;
   MEDIA_UNION rxU;

   // Transmit Info
   H324_BOOL    bTransmit;
   int          txChannel;
   DWORD        txChoice;
   MEDIA_UNION txU;
}H324_LCD;
```

Fields in the H324_LCD structure

Field name	Description
bReceive	Indicates whether the rxU field contains media options for the Rx channel in a full-duplex configuration. Valid values are: TRUE - rxU field contains media options for the Rx channel. FALSE - rxU field is not in use.
rxChannel	The channel number of the receiving channel.
rxChoice	Describes how to interpret rxU union. Valid values are: H324_MPEG4_VIDEO (0) H324_H263_VIDEO (1) H324_H261_VIDEO (2) (Not supported) H324_G711_AUDIO (3) (Not supported) H324_G723_AUDIO (4) H324_AMR_AUDIO (5) H324_H264_VIDEO (6)
rxU	MEDIA_UNION structure, which contains different media options, according to the value of the txChoice field.
bTransmit	Whether the txU field contains media options for the Tx channel. Valid values are: TRUE: txU field contains media options for the Tx channel. FALSE: txU field is not in use.

Field name	Description
txChannel	Channel number of the channel that transmits data.
txChoice	Describes how to interpret txU union. Valid values are: H324_MPEG4_VIDEO (0) H324_H263_VIDEO (1) H324_H261_VIDEO (2) (Not supported) H324_G711_AUDIO (3) (Not supported) H324_G723_AUDIO (4) H324_AMR_AUDIO (5) H324_H264_VIDEO (6)
txU	MEDIA_UNION structure, which contains different media options, according to the value of the txChoice field.

Fields in the MEDIA_UNION structure

Type	Field name	Description
MMP4VIDEOOPTIONS	mpeg4	MMP4VIDEOOPTIONS structure that contains MPEG-4 video options.
MH263OPTIONS	h263	MH263OPTIONS structure that contains H.263 video options.
MH261OPTIONS	h261	H.261 video options. (Not supported)
MG711OPTIONS	g711	MG711OPTIONS structure that contains G.711 audio options. (Not supported)
MG7231OPTIONS	g723	MG7231OPTIONS structure that contains G.723.1 audio options.
MGSMAMROPTIONS	gsmamr	MGSMAMROPTIONS structure that contains AMR audio options.
H264VIDEOOPTIONS	h264	H264VIDEOOPTIONS structure that contains H.264 video options.
MAMRWBOPTIONS	amrwb	MAMRWBOPTIONS structure that contains AMR-WB video options. (Not supported)
USERINPUTOPTIONS	uinp	The USERINPUTOPTIONS structure that tells the remote side what types of user input messages the 3G-324M Interface can decode. The 3G-324M Interface currently supports all user interface options except for option 11 (generic user input).

The USERINTPUTOPTIONS structure definition is:

```
#define MAXSUBIDS 128
struct _h245ObjIdentifier
{
unsigned int     numOfids;
unsigned int     subid[MAXSUBIDS];
};struct _H245_H221NonStandard
{
unsigned char  countryCode;        //t35
unsigned char  extension;          //t35
unsigned short manufacturerCode;   //0...65535
};
struct _H245_NonStandardIdentifier   //of type h221 non standard, refer to annex B of
rec.
{
#define H245_OBJECT         1
#define H245_NON_STANDARD   2
char choice;
union
{
t_H245ObjIdentifier* object;
t_H245_H221NonStandard* h221NonStandard;
}u;
};typedef struct _USERINPUTOPTIONS
{
  #define USER_INPUT_NON_STANDARD  0;
  #define USER_INPUT_BASIC_STRING  1;
  #define USER_INPUT_IA5_STRING  2;
  #define USER_INPUT_GENERAL_STRING  3;
  #define USER_INPUT_DTMF  4;
  #define USER_INPUT_HOOKFLASH  5;
  #define USER_INPUT_EXTENDED_ALPHA_NUMERIC  6;
  #define USER_INPUT_ENCRYPTED_BASIC_STRING  7;
  #define USER_INPUT_ENCRYPTED_IA5_STRING  8;
  #define USER_INPUT_ENCRYPTED_GENERAL_STRING  9;
  #define USER_INPUT_SECURE_DTMF 10;
  #define USER_INPUT_GENERIC_USER_INPUT_CAPABILITY 11;
  int choice;
unsigned char nonStandardCount;
t_H245_NonStandardParameter nonStandardParam[16];
} USERINPUTOPTIONS;
```

For detailed information about the codec-related structures in MEDIA_UNION, see the associated ITU-T specifications.

H324_START_PARAMS

The host application uses this structure to initialize the H.245 stack for the specified MUX endpoint. This structure is used with **h324Start**.

Definition

```
typedef struct tag_H324_START_PARAMS
{
   DWORD        size;
   int          terminalType;
   int          iCapCount;
   int          iCap[16];
   H324_BOOL    bAutoChannelSetup;
} H324_START_PARAMS;
```

Fields

Field name	Description
size	Size of H324_START_PARAMS structure.
terminalType	Terminal type of the local terminal. The H.245 stack makes a decision about master/slave status by comparing the local and remote terminalType values. Usually, terminals have a terminal type value of 128.
iCapCount	Number of entries used in the iCap (capability table) array. Valid values are integers from 1 to 16.
iCap[16]	List of capabilities that the host application supports. Valid values are: H324_MPEG4_VIDEO (0) H324_H263_VIDEO (1) H324_H261_VIDEO (2) (Not supported) H324_G711_AUDIO (3) (Not supported) H324_G723_AUDIO (4) H324_AMR_AUDIO (5) H324_H264_VIDEO (6) H324_RX_USER_INPUT (8)
bAutoChannelSetup	Not applicable.

H324_TERM_CAPS

Conveys information about terminal capability sets between the host application and the remote terminal. The H324_TERM_CAPS structure resides in a set of structures that also includes:

Structure	Description
MULTIPLEX_CABILITY	Describes the MUX capabilities of the local or remote terminal.
MEDIA_CAPABILITY	Describes the media capabilities of the local or remote terminal.
MEDIA_UNION	Contains media options for specific media types.

The H324_TERMCAPS structure set is used as follows:

- The host application uses this structure set to send a list of its terminal capabilities to the remote terminal. In this situation, the structure set is used with **h324SetTermCaps**.
- The remote terminal uses this structure set to send a list of its terminal capabilities to the host application. In this situation, the structure set is used with H324EVN_REMOTE_CAPABILITIES events.

Definitions

```
typedef struct tag_MULTIPLEX_CAPABILITY
{
   H324_BOOL                        dataWithAL1;
   H324_BOOL                        audioWithAL2;
   H324_BOOL                        videoWithAL2;
   H324_BOOL                        videoWithAL3;
   WORD                             maximumAL1MPDUSize;
   WORD                             maximumAL2MSDUSize;
   WORD                             maximumAL3MSDUSize;
} MULTIPLEX_CAPABILITY;

typedef union tag_MEDIA_UNION
{
   MMP4VIDEOOPTIONS                 mpeg4;
   MH263OPTIONS                     h263;
   MH261OPTIONS                     h261;     (Not supported)
   MG711OPTIONS                     g711;     (Not supported)
   MG7231OPTIONS                    g723;
   MGSMAMROPTIONS                   gsmamr;
   H264VIDEOOPTIONS                 h264;
   MAMRWBOPTIONS                    anrwb;    (Not supported)
   USERINPUTOPTIONS                 uinp;
} MEDIA_UNION;

typedef struct tag_MEDIA_CAPABILITY
{
   H324_BOOL                        bReceive;
   H324_BOOL                        bTransmit;
   int                              index;
   DWORD                            choice;
#define H324_MPEG4_VIDEO    0
#define H324_H263_VIDEO     1
#define H324_H261_VIDEO     2                (Not supported)
#define H324_G711_AUDIO     3                (Not supported)
#define H324_G723_AUDIO     4
#define H324_AMR_AUDIO      5
#define H324_H264_VIDEO 6
   DWORD                    size;
   MEDIA_UNION u;
} H324_MEDIA_CAPABILITY;

typedef struct tag_H324_TERM_CAPS
{
   DWORD                            size;
   MULTIPLEX_CAPABILITY             muxCap;
   DWORD                            wCapCount;  // Number of caps
   H324_MEDIA_CAPABILITY            capTable[16];
} H324_TERM_CAPS;
```

Fields in the MULTIPLEX_CAPABILITY structure

Field name	Description
dataWithAL1	Whether the originating entity supports data transmission with adaptation layer 1. (Not supported)
audioWithAL2	Whether the originating entity supports audio transmission with adaptation layer 2.
videoWithAL2	Whether the originating entity supports video transmission with adaptation layer 2.
videoWithAL3	Whether the originating entity supports video transmission with adaptation layer 3.
maximumAL1MPDUSize	Maximum size in octets for the PDUs that the multiplexer can receive in adaptation layer 1. (Not supported)
maximumAL2MSDUSIZE	Maximum size in octets for the SDUs that the multiplexer can receive in adaptation layer 2.
maximumAL3MSDUSIZE	Maximum size in octets for the SDUs that the multiplexer can receive in adaptation layer 3.

Fields in the MEDIA_UNION structure

Type	Field name	Description
MMP4VIDEOOPTIONS	mpeg4	MMP4VIDEOOPTIONS structure that contains MPEG-4 video options.
MH263OPTIONS	h263	MH263OPTIONS structure that contains H.263 video options.
MH261OPTIONS	h261	H.261 video options (Not supported)
MG711OPTIONS	g711	MG711OPTIONS structure that contains G.711 audio options. (Not supported)
MG7231OPTIONS	g723	MG7231OPTIONS structure that contains G.723.1 audio options.
MGSMAMROPTIONS	gsmamr	MGSMAMROPTIONS structure that contains AMR audio options
H264VIDEOOPTIONS	h264	H264VIDEOOPTIONS structure that contains H.264 video options.
MAMRWBOPTIONS	amrwb	MAMRWBOPTIONS structure that contains AMR-WB video options. (Not supported)
USERINPUTOPTIONS	uinp	USERINPUTOPTIONS structure tells the remote side what types of user input messages the 3G-324M Interface can decode. The 3G-324M Interface currently supports all user interface options except for option 11 (generic user input).

Fields in the MEDIA_CAPABILITY structure

Field name	Description
bReceive	Whether this capability is for receiving data.
bTransmit	Whether this capability is for transmitting data.
index	Entry number for this capability.
choice	One of the capabilities that the originating entity supports. Valid values are: H324_MPEG4_VIDEO or 0 H324_H263_VIDEO or 1 H324_H261_VIDEO or 2 (Not supported) H324_G711_AUDIO or 3 (Not supported) H324_G723_AUDIO or 4 H324_AMR_AUDIO or 5 H324_H264_VIDEO or 6
size	Size of the MEDIA_CAPABILITY structure.
u	MEDIA_UNION structure that contains different media options, according to the value of the choice field.

The USERINTPUTOPTIONS structure definition is:

```
typedef struct _USERINPUTOPTIONS
{
#define USER_INPUT_NON_STANDARD  0;
#define USER_INPUT_BASIC_STRING  1;
#define USER_INPUT_IA5_STRING  2;
#define USER_INPUT_GENERAL_STRING  3;
#define USER_INPUT_DTMF  4;
#define USER_INPUT_HOOKFLASH  5;
#define USER_INPUT_EXTENDED_ALPHA_NUMERIC  6;
#define USER_INPUT_ENCRYPTED_BASIC_STRING  7;
#define USER_INPUT_ENCRYPTED_IA5_STRING  8;
#define USER_INPUT_ENCRYPTED_GENERAL_STRING  9;
#define USER_INPUT_SECURE_DTMF 10;
#define USER_INPUT_GENERIC_USER_INPUT_CAPABILITY 11;
int choice;
unsigned char nonStandardCount;
t_H245_NonStandardParameter nonStandardParam[16];
} USERINPUTOPTIONS;
```

For detailed information about the codec-related structures in MEDIA_UNION, see the associated ITU-T specifications.

Fields in the H324_TERM_CAPS structure

Field name	Description
size	Size of the H324_TERM_CAPS structure.
muxCap	List of multiplexer capabilities that the originating entity can perform, presented in the MULTIPLEX_CAPABILITY structure.
wCapCount	Number of entries used in the capTable array. Valid values are integers from 1 to 16.
capTable[16]	Array of media capabilities that the originating entity can perform, presented in the MEDIA_CAPABILITY structure.

H324_USER_INPUT_INDICATION

The remote terminal uses this structure to send a user indication message to the host application. In this situation, the structure is used with inbound H324EVN_USER_INDICATION events.

The host application uses this structure to send a user indication message to the remote terminal. In this case, the structure is used with **h324UserIndication**.

Definition

```
typedef struct tag_H324_USER_INPUT_INDICATION
{`
   char     data[H324_MAX_USER_INDICATION_SIZE];
   DWORD    length;
   char     szObjectId[H324_MAX_UII_OBJECT_ID_SIZE];
   DWORD    msgType;
} H324_USER_INPUT_INDICATION;
```

Fields

Field name	Description
data[H324_MAX_USER_INDICATION_SIZE]	User indication data (usually one or more keystrokes), if the message type is alphanumeric or signal. The size of H324_MAX_USER_INDICATION_SIZE is 512 bytes.
length	Number of keystrokes in the data field.
szObjectId[H324_MAX_UII_OBJECT_ID_SIZE]	Object identifier for a nonstandard user input indication. Set this identifier by copying a zero terminated string into this field. The string is a dot-separated number sequence, such as 1.2.123.2345.5.73. The size for H324_MAX_UII_OBJECT_ID_SIZE] is 128 bytes. If the message type is alphanumeric or signal, then this field is set to 0.
msgType	Type of user input indication that you are sending to the remote terminal. Valid values are: H324_USER_INPUT_ALPHANUMERIC - Sending a string of one or more characters. H324_USER_INPUT_NONSTANDARD - Sending an object identifier in the szObjectID field. H324_USER_INPUT_SIGNAL - Sending an H.245 signal consisting of a DTMF key press. The signal can be one of the following digits (0123456789*#ABCD) or a hookflash (!).

H324_VENDORID_INDICATION

The remote terminal uses this structure to send vendor information to the host application. In this situation, the structure is used with inbound H324EVN_VENDOR_INDICATION events.

The host application uses this structure to send vendor information to the remote terminal. In this case, the data structure is used with **h324VendorIDIndication**.

In the H324_VENDORID_INDICATION structure, the vendorID field is mandatory but the productNumber and versionNumber fields are optional, making this a variable-length structure.

Definition

```
typedef struct tag_H324_VENDORID_INDICATION
{
   unsigned short    vendorIDLen;
   unsigned short    productNumberLen;
   unsigned short    versionNumberLen;
   unsigned char     isNonStandard;
   unsigned char     bytes[sizeof(char)];
   // "vendorID" field resides at bytes[0] through bytes[vendorIDLen-1].
   // OPTIONAL "productNumber" resides at bytes[vendorIDLen]
   //          through bytes[vendorIDLen+productNumberLen-1].
   // OPTIONAL "versionNumber" resides at bytes[vendorIDLen+productNumberLen]
   //          through bytes[vendorIDLen+productNumberLen+versionNumberLen-1].
} H324_VENDORID_INDICATION;
```

Fields

Field name	Description
vendorIDLen	Length in bytes of the vendorID field.
productNumberLen	Length in bytes of the productNumber field.
versionNumberLen	Length in bytes of the versionNumber field.
isNonStandard	Specifies the type of vendor ID. Valid values are: • NZ - Vendor ID is of type h221NonStandard. • 0 - VendorID field contains an Object ID.
vendorID	Vendor ID for the terminal or component. The value of this field depends on the value of the isNonStandard field. • If the value of the isNonStandard field is NZ, then the vendor ID field is of type h221NonStandard. • If the value of the isNonStandard field is false or 0, then the vendor ID field contains an object ID.
productNumber	Product number for the terminal or component. This field is present only if productNumberLen is not set to 0.
versionNumber	Version number for the terminal or component. This field is present only if versionNumberLen is not set to 0.

H324_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION

The remote terminal uses this structure to pass temporal-spatial tradeoff indications to the host application. This structure is used with inbound H324EVN_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION events.

Definition

```
typedef struct tag_H324_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION
{
   unsigned int    logicalChannelNumber;
   unsigned int    tradeoff;
} H324_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION;
```

Fields

Field name	Description
logicalChannelNumber	Logical channel number of the delayed audio or video data.
tradeoff	Temporal-spatial tradeoff value for the specified channel number. Tradeoff values range from 0 to 31, with a tradeoff value of 0 indicating a high spatial resolution and a tradeoff value of 31 indicating a high frame rate. Actual values do not correspond to precise values of spatial resolution or frame rate.

12 H.324M events and reason codes

Working with H.324M events

The host application must submit all events received from Natural Access to the H.324M Middleware by invoking h324SubmitEvent. The Middleware processes these events, and then returns them to the application when relevant.

The Middleware can either consume or not consume each event submitted to it. If an event is:

- Consumed, the application continues to wait for additional Natural Access events.
- Not consumed, it is an event for the application, which should process it normally.

If an unconsumed event has an attached buffer, the application needs to free the buffer using either **ctaFreeBuffer**, or **mspReleaseBuffer** as appropriate. The event size field defines the size of the buffer in bytes.

For more information, see *h324SubmitEvent* on page 129 and the *Natural Access Developer's Reference Manual*.

Alphabetical event summary

This following table describes the H.324M events that can be sent by the H.324M Middleware to the application:

Event	Description
H324EVN_CALL_SETUP_FAILED	Call setup failed due to a failure of either the master slave determination process or the terminal capabilities exchange process. One of the following reason codes is returned: H324RSN_TERM_CAP_REJECT_UNSPECIFIED H324RSN_TERM_CAP_REJECT_UNDEFINED_TABLE _ENTRY_USED H324RSN_TERM_CAP_REJECT_DESCRIPTOR _CAPACITY_EXCEEDED H324RSN_TERM_CAP_REJECT_TABLE_ENTRY _CAPACITY_EXCEEDED H324RSN_TERM_CAP_TIME_OUT H324RSN_MASTER_SLAVE_ERROR H324RSN_UNSPECIFIED Shut down the call by calling **h324Stop** and **h324Delete**.
H324EVN_CHANNEL_CLOSED	Either the application or the remote terminal successfully closed a channel. The type of the channel closed is indicated in the event size field, which contains one of the following values: H324RSN_AUDIO_IN H324RSN_AUDIO_OUT H324RSN_VIDEO H324RSN_VIDEO_IN H324RSN_VIDEO_OUT
H324EVN_CHANNEL_CLOSE_FAILED	Remote terminal rejected the request to close the specified channel.
H324EVN_END_SESSION	Remote terminal issued an H.245 EndSession command, which initiates call teardown. Once this command is issued, the only calls the H.324M Middleware expects are **h324Stop** and **h324Delete**. The Middleware also does not expect any further incoming H.324 events, except for H324EVN_STOP_DONE. **Note:** Many terminals do not send H324EVN_END_SESSION prior to terminating the call.
H324EVN_END_SESSION_DONE	End session message was delivered to the remote terminal.
H324EVN_END_SESSION_TIMER_EXPIRED	Internal event that normally gets consumed by **h324SubmitEvent**. If the event is not consumed, the application must ignore it. The event may not be consumed if the corresponding instance of the H.245 stack was destroyed before submitting the event to the H.324M Middleware.

Event	Description
H324EVN_FAST_CALL_SETUP_DONE	Call was set up using MONA (Media Oriented Negotiation Acceleration Procedures), which is defined in Annex K/H.324. For more information, see *Enabling fast call setup* on page 23.
H324EVN_H223_SKEW_INDICATION	H.223 skew indication message from the remote terminal. This message indicates that one channel (either audio or video) is delayed with respect to the other. The application must apply a corresponding delay to the other channel to resynchronize playback of the two channels. The 3G-324M Interface does not provide a resynchronization mechanism. The event buffer contains this message in a structure of type H324_H223_SKEW_INDICATION.
H324EVN_H245_INTERNAL_ERROR	H.245 stack declared an internal error of an undefined type. The recommended resolution is to terminate the call.
H324EVN_INTERNAL_EVENT	Internal event that gets consumed by **h324SubmitEvent**.
H324EVN_LCD	One of the following: • The H.324M Middleware accepted a unidirectional remote OLC request from the remote terminal. • The H.324M Middleware received a confirmation in the case of a bi-directional OLC from the remote terminal. • The remote terminal accepted a unidirectional OLC sent by H.324M Middleware. The contents of this set are reflected in the event buffer as a structure of type H324_LCD.
H324EVN_MEDIA_SETUP_DONE	All of the following conditions have been met: • Two unidirectional audio channels were opened. • One bi-directional or two unidirectional video channels were opened. • Multiplex table exchanges were completed.
H324EVN_MEDIA_SETUP_FAILED	Media setup failed due to a failure of either the multiplex entry table, or one of the audio channels. One of the following reason codes is returned: H324RSN_MUX_TABLE_REJECT_UNSPECIFIED H324RSN_MUX_TABLE_REJECT_DESCRIPTOR_TOO_COMPLEX H324RSN_INBOUND_AUDIO_CHANNEL_FAILURE H324RSN_OUTBOUND_AUDIO_CHANNEL_FAILURE H324RSN_LCSE_ERROR_INDICATION H324RSN_UNSPECIFIED Shut down the call by calling **h324Stop** and **h324Delete**.
H324EVN_PASSTHRU_DTMF_MODE_DONE	DTMF mode and payload ID were reset.

Event	Description
H324EVN_PASSTHRU_PLAY_RFC2833_DONE	RFC 2833 DTMF digits are passed to the IP side as RTP-encapsulated packets.
H324EVN_REMOTE_CAPABILITIES	Message from the remote terminal that conveys its set of terminal capabilities. Event buffer contains this information in a structure of type H324_TERM_CAPS.
H324EVN_ROUND_TRIP_DELAY	Results of round trip delay request.
H324EVN_ROUND_TRIP_TIMEOUT	Timeout after a round trip delay message was sent.
H324EVN_START_DONE	Received when the H.245 stack creation is complete. The event value indicates success or failure. One of the following reason codes can be returned: CTA_REASON_FINISHED – Returned if the call to **h324Start** was successful. CTA_REASON_TIMEOUT – Returned if there was a H.223 level synchronization problem with the remote terminal.
H324EVN_START_TIMER_EXPIRED	Internal event to be consumed by **h324SubmitEvent**. If the H324EVN_START_TIMER_EXPIRED event is not consumed, the application must ignore it. The event is not consumed if the corresponding instance of the H.245 stack was destroyed before submitting the event to the H.324M Middleware.
H324EVN_STOP_DONE	Received when the H.245 stack destruction is complete. The event value indicates success or failure. The application must call **h324Delete** when this event is received.
H324EVN_USER_INDICATION	User indication message from the remote terminal. The event buffer contains this message in a structure of type H324_USER_INPUT_INDICATION.
H324EVN_VENDORID_INDICATION	Vendor indication message from the remote terminal. The event buffer contains this message in a structure of type H324_VENDORID_INDICATION.
H324EVN_VIDEO_CHANNEL_SETUP_FAILED	After multiple retries, the H.324M Middleware could not establish a bi-directional video channel. The call can continue as an audio only call, or can be shut down by calling **h324Stop** and **h324Delete**.
H324EVN_VIDEO_FAST_UPDATE	Request from the remote terminal for a video fast update.
H324EVN_VIDEO_OLC_TIMER_EXPIRED	Internal event to be consumed by **h324SubmitEvent**. If the event is not consumed, the application must ignore it. The event may not be consumed if the corresponding instance of the H.245 stack was destroyed before submitting the event to the H.324M Middleware.
H324EVN_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION	Message from the remote terminal that conveys the temporal-spatial tradeoff indication for the terminal's outbound video channel. The event buffer contains this information in a structure of type H324_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION.

Numerical event summary

The following table numerically lists the H.324M events that can be sent by the H.324M Middleware to the application:

Hex	Decimal	Event
0x522000	5382144	H324EVN_START_DONE
0x522001	5382145	H324EVN_LOCAL_TERMCAPS
0x522003	5382147	H324EVN_REMOTE_CAPABILITIES
0x522005	5382149	H324EVN_MEDIA_SETUP_DONE
0x522006	5382150	H324EVN_VIDEO_FAST_UPDATE
0x522007	5382151	H324EVN_END_SESSION
0x522008	5382152	H324EVN_USER_INDICATION
0x522009	5382153	H324EVN_ROUND_TRIP_DELAY
0x52200A	5382154	H324EVN_ROUND_TRIP_DELAY
0x52200B	5382155	H324EVN_STOP_DONE
0x52200F	5382159	H324EVN_H245_INTERNAL_ERROR
0x522010	5382160	H324EVN_CALL_SETUP_FAILED
0x522011	5382161	H324EVN_MEDIA_SETUP_FAILED
0x522012	5382162	H324EVN_VIDEO_CHANNEL_SETUP_FAILED
0x522013	5382164	H324EVN_CHANNEL_CLOSED
0x522014	5382164	H324EVN_VIDEO_OLC_TIMER_EXPIRED
0x522015	5382165	H324EVN_END_SESSION_TIMER_EXPIRED
0x522016	5382166	H324EVN_END_SESSION_DONE
0x522017	5382167	H324EVN_H223_SKEW_INDICATION
0x522018	5382168	H324EVN_VENDORID_INDICATION
0x522019	5382169	H324EVN_VIDEOTEMPORALSPATIALTRADEOFF_INDICATION
0x52201A	5382170	H324EVN_PASSTHRU_PLAY_RFC2833_DONE
0x52201B	5382171	H324EVN_PASSTHRU_DTMF_MODE_DONE
0x52201C	5382172	H324EVN_LCD
0x52201D	5382173	H324EVN_START_TIMER_EXPIRED
0x52201E	5382174	H324EVN_CHANNEL_CLOSE_FAILED
0x522021	5382177	H324EVN_FAST_CALL_SETUP_DONE

Reason codes

The following table lists the 3G-324M Interface reason codes in alphabetical order:

Reason code	Hex	Decimal	Description
H324RSN_AUDIO_IN	0x000020	0000032	Audio inbound channel identifier.
H324RSN_AUDIO_OUT	0x000021	0000033	Audio outbound channel identifier.
H324RSN_INBOUND_AUDIO_CHANNEL_FAILURE	0x00000B	0000011	Failure to establish audio on an inbound channel.
H324RSN_LCSE_ERROR_INDICATION	0x00000D	0000013	Indication of an OLC error sent by the remote terminal.
H324RSN_MASTER_SLAVE_ERROR	0x000007	0000007	Master/slave determination procedure failed.
H324RSN_MUX_TABLE_REJECT_DESCRIPTOR_TOO_COMPLEX	0x000009	0000009	Multiplex table was rejected by the remote terminal. The MultiplexEntryDescriptor exceeded the capability of the receive terminal.
H324RSN_MUX_TABLE_REJECT_UNSPECIFIED	0x000008	0000008	Multiplex table was not specified.
H324RSN_MUX_TABLE_TIME_OUT	0x00000A	0000010	Timeout occurred during the multiplex tables exchange procedure.
H324RSN_OUTBOUND_AUDIO_CHANNEL_FAILURE	0x00000C	0000012	Failure to establish audio on an outbound channel.
H324RSN_TERM_CAP_ERROR_UNSPECIFIED	0x000006	0000006	Could not send the terminal capabilities set to the remote terminal.
H324RSN_TERM_CAP_REJECT_DESCRIPTOR_CAPACITY_EXCEEDED	0x000002	0000002	Remote terminal could not store the information in the TerminalCapabilitySet.
H324RSN_TERM_CAP_REJECT_TABLE_ENTRY_CAPACITY_EXCEEDED	0x000003	0000003	Remote terminal could not store more entries than indicated in highestEntryNumberProcessed or could not store any entries.
H324RSN_TERM_CAP_REJECT_UNDEFINED_TABLE_ENTRY_USED	0x000001	0000001	Capability descriptor made reference to a capability table entry that is not defined.
H324RSN_TERM_CAP_REJECT_UNSPECIFIED	0x000000	0000000	No cause for rejection was specified by the remote terminal.
H324RSN_TERM_CAP_REMOTE_REJECT	0x000005	0000005	Terminal capability set was rejected.
H324RSN_TERM_CAP_TIME_OUT	0x000004	0000004	Timeout occurred during terminal capabilities exchange procedure.

Reason code	Hex	Decimal	Description
H324RSN_UNSPECIFIED	0x00000E	0000014	Unspecified channel identifier.
H324RSN_VIDEO	0x000022	0000034	Bidirectional video channel identifier.
H324RSN_VIDEO_IN	0x000023	0000035	Incoming unidirectional video channel identifier.
H324RSN_VIDEO_OUT	0x000024	0000036	Outgoing unidirectional video channel identifier.

13 H.324M error codes

Working with H.324M error codes

All functions return a status code. If the return code is not SUCCESS (0), it is an error code indicating that the function failed and the reason for the failure.

H.324M Middleware error codes are defined in the *h324def.h* include file. The error codes are prefixed with H324ERR.

For errors beginning with CTAERR, refer to the *Natural Access Developer's Reference Manual*.

Alphabetical error summary

The following table describes H.324M Middleware errors. All errors are 32 bits.

Error	Hex	Decimal	Description
H324ERR_ALREADY_INITIALIZED	0x520001	5373953	**h324Initialize** was already called.
H324ERR_COMMAND_RESPONSE_ERROR	0x520008	5373960	A command done event received from an unexpected command.
H324ERR_INCOMING_MSG_ERROR	0x520007	5373959	Error processing an incoming H.245 message.
H324ERR_INITIALIZE_STACK_FAILED	0x520006	5373958	Unable to initialize the H.245 stack.
H324ERR_INTERNAL_ERROR	0x520005	5373957	Internal error in the H.324M Middleware.
H324ERR_LOG_FILE_OPEN_FAILED	0x52000B	5373963	Error opening the log file.
H324ERR_MUTEX_CREATE_FAILED	0x520002	5373954	Failed to create mutex semaphore.
H324ERR_MUTEX_LOCK_FAILED	0x520004	5373956	Internal error trying to lock mutex.
H324ERR_NOT_INITIALIZED	0x520003	5373955	H.324M Middleware is not initialized. Call **h324Initialize** first.
H324ERR_OUT_OF_MEMORY	0x52000A	5373962	Failed to allocate memory.
H324ERR_PMSYNC_ERROR	0x520009	5373961	Error after multiplexer synchronizes with remote multiplexer.

Numerical error summary

The following table lists 3G-324M Interface errors in numerical order:

Hex	Decimal	Error
0x520001	5373953	H324ERR_ALREADY_INITIALIZED
0x520002	5373954	H324ERR_MUTEX_CREATE_FAILED
0x520003	5373955	H324ERR_NOT_INITIALIZED
0x520004	5373956	H324ERR_MUTEX_LOCK_FAILED
0x520005	5373957	H324ERR_INTERNAL_ERROR
0x520006	5373958	H324ERR_INITIALIZE_STACK_FAILED
0x520007	5373959	H324ERR_INCOMING_MSG_ERROR
0x520008	5373960	H324ERR_COMMAND_RESPONSE_ERROR
0x520009	5373961	H324ERR_PMSYNC_ERROR
0x52000A	5373962	H324ERR_OUT_OF_MEMORY
0x52000B	5373963	H324ERR_LOG_FILE_OPEN_FAILED

Index