Dialogic®

Dialogic® 3G-324M API

Programming Guide and Library Reference

June 2011

05-2558-006

Publication Date: June 2011

Document Number: 05-2558-006

Contents

Figures

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2558-006	June 2011	Removed references to Dialogic® Multimedia Software for AdvancedTCA and Dialogic® Multimedia Kit Software for PCIe as these products are no longer supported. m3g_EnableEvents( ) function: Added M3GEV_IFRAME_RCVD to list of unsolicited events and M3G_IFRAME_EVT_TYP bit mask to parameter table. m3g_GetLocalCaps( ) function: Added H.264 to video device. Events chapter: Added M3GEV_IFRAME_RCVD for H.264 support. M3G_H264_OPTIONS data structure: New. M3G_IFRAME_DATA data structure: New. M3G_PARM_INFO data structure: Updated structure definition. Added M3G_E_PRM_IFRAME_NOTIFY_CONTROL_MASK and M3G_E_PRM_H264_TX_DCI parameter types. M3G_VIDEO_CAPABILITY data structure: Added M3G_E_H264 to coderType field. M3G_VIDEO_OPTIONS data structure: Added h264 field.
05-2558-005	October 2008	Added programming guide content to create a combined Programming Guide and Library Reference. Programming Guide content: Added the following chapters: Product Description, Device Handling, Event Handling, Error Handling, Implementing a 3G-324M Session, Interoperability and Compliance Information, Video Quality Considerations, Data Structure Considerations, Building Applications, and Debugging. Device Handling chapter: Added a note to board device that functions called on a board device affect all channels on that device. [IPY00078711] m3g_EnableEvents( ) and m3g_DisableEvents( ) functions: Added maskable events, M3GEV_MONA_PREF_MSG_RCVD, M3GEV_SEND_MONA_PREF_MSG, and M3GEV_CALL_STATISTICS. Added M3G_MONA_PREF_MSG_EVT_TYP and M3G_CALL_STATISTICS_EVT_TYP to eventBitMask. m3g_ModifyMedia( ) function: added information about when to use this function. m3g_OpenLC( ) function: Updated example code for MONA support. m3g_Reset( ) function: Clarified use of this function. m3g_SetVendorId( ) function: Added statement that changes to vendor ID are in effect until services are restarted. m3g_StartH245( ) function: Updated third paragraph to state that this function should be called after a control device is opened (rather than audio/video devices). Added information about MONA. Updated example code for MONA support. m3g_StartMedia( ) function: Updated second paragraph to include MONA MPCs. m3g_StartTrace( ) function: Added default location for logfiles. m3g_StopH245( ) function: Updated second paragraph due to MONA support.

Document No.	Publication Date	Description of Revisions
05-2558-005 (continued)	October 2008	Events chapter: Corrected data type for M3GEV_REMOTE_VENDORID_RCVD (M3G_VENDORID_INFO not M3G_VENDOR_INFO). Added new events for MONA support: M3GEV_MONA_PREF_MSG_RCVD, M3GEV_TX_MPC_ESTABLISHED, M3GEV_RX_MPC_ESTABLISHED, M3GEV_SEND_MONA_PREF_MSG, and M3GEV_CALL_STATISTICS. Data Structures chapter: Added the following new structures: M3G_CALL_STATISTICS, M3G_H221_NONSTD, M3G_MONA_MPC, M3G_MONA_TXRX_MPC_SUPPORT, M3G_NONSTANDARD_ID, M3G_OBJECT_ID. M3G_CAPS_LIST data structure: Expanded description to state that capabilities are listed in decreasing order of preference. M3G_H223_SESSION data structure: Added isMONAEnabled for MONA support. M3G_PARM_INFO data structure: Added M3G_E_PRM_EARLY_MES, M3G_E_PRM_AUTO_VFU_PERIOD, M3G_E_PRM_H223_SYNC_TIMER. M3G_TRACE_INFO data structure: Added default location for logfiles. Added more detail to bitmask descriptions.
05-2558-004	April 2008	Function Summary by Category chapter: Added Utility Functions category for 3G-324M tracing feature. Added **m3g_SetVendorId( )** to H.245 Control Functions. m3g_EnableEvents( ) function: Added M3G_REMOTE_VENDORID_RCVD event and M3G_REMOTE_VENDORID_EVT_TYP bitmask. m3g_ModifyMedia( ) function: Updated M3GEV_MODIFY_MEDIA_CMPLT event description to show support for **sr_getevtdatap( )** function. Updated Example code "case M3GEV_MODIFY_MEDIA_CMPLT" to reflect this change. m3g_SetVendorId( ) function: New. Added for vendor ID feature. m3g_StartTrace( ) function: New. Added for 3G-324M tracing feature. m3g_StopTrace( ) function: New. Added for 3G-324M tracing feature. Events chapter: Added M3GEV_START_TRACE_CMPLT, M3GEV_START_TRACE_FAIL, M3GEV_STOP_TRACE_CMPLT, M3GEV_STOP_TRACE_FAIL, M3GEV_SET_VENDORID_CMPLT, M3GEV_SET_VENDORID_FAIL, M3GEV_REMOTE_VENDORID_RCVD. Data Structures chapter: Added the following new data structures: M3G_TRACE_INFO and M3G_VENDORID_INFO. Data Structures chapter: Corrected the data type of the version field in all applicable structures from unsigned short to unsigned int (M3G_AUDIO_CAPABILITY, M3G_CAPS_LIST, M3G_H223_SESSION, M3G_H245_MISC_CMD, M3G_H245_UII, M3G_REMOTE_CLOSED_LC, M3G_REMOTE_OLC_REQ, M3G_REMOTE_OLCACK_RESP, M3G_START_STRUCT, M3G_VIDEO_CAPABILITY). Data Structures chapter: Updated the description of the version field in all applicable structures to refer to the symbolic constant M3G_LIBRARY_VERSION. M3G_H223_LC_PARAMS structure: Added INIT inline function. M3G_H223_SESSION structure: Added INIT inline function. M3G_H245_CMD structure: Added INIT inline function. M3G_H245_UII structure: Added INIT inline function. M3G_PARM_INFO structure: Added INIT inline function. Corrected misspelling ("FAST**UPD**ATE" rather than "FAST**UDP**ATE") in M3G_E_PRM_RELAY_FASTUPDATE_TO_MEDIA_DEV and M3G_E_PRM_RELAY_FASTUPDATE_TO_H245.

Document No.	Publication Date	Description of Revisions
05-2558-003	February 2008	m3g_ModifyMedia( ) function: Updated description (does not support mute or resume channels.) Updated example code. M3G_PARM_INFO data structure: Updated with new fields (skewAdjustment, videoBitRate, videoFrameRate). Added parameter types (M3G_E_PRM_TX_SKEW_ADJUSTMENT, M3G_E_PRM_RX_SKEW_ADJUSTMENT, M3G_E_PRM_VIDEO_BIT_RATE, M3G_E_PRM_VIDEO_FRAME_RATE) to Table 1. Added Data Type column to this table.
05-2558-002	October 2007	MPEG-4 transcoding is now supported. m3g_GetLocalCaps( ) and m3g_GetMatchedCaps( ) functions: Updated video description to add MPEG-4 capabilities. m3g_Open( ) function: Updated the audio device description to include statement that it may be connected to a digital network interface device (dtiBxTy) or voice device (dxxxBxCy) through the **dev_Connect( )** and **dev_Disconnect( )** functions in Device Types section. m3g_StartMedia( ) function: Updated the description to include statement that PCM network device must be connected via the **dev_Connect( )** function prior to calling this function in Description section. m3g_StopH245( ) function: Updated the description to include **dev_Disconnect( )** as an option when disconnecting the audio and video devices in Description section. Events chapter: Updated the M3GEV_REMOTE_CLOSE_LC_RCVD event description to include **dev_Disconnect( )** as an option when re-routing the associated media stream from the H.223 aggregrate and stopping the media stream in Event Information section. Data Structures chapter: Added the following new data structure: M3G_OCTET_STRING. M3G_MPEG4_OPTIONS structure: Updated the structure declaration with comments. Updated the decoderConfigLength and decoderConfigInfo fields with more details in Field Descriptions section. Changed the visualBackChannel field to state that Default value is M3G_FALSE in Field Descriptions section. M3G_PARM_INFO structure: Updated the structure declaration with octetString field. Added the octetString field along with description in Field Descriptions section. Added the M3G_E_PRM_MPEG4_TX_DCI and M3G_E_PRM_MPEG4_RX_DCI parameters to Table 1.
05-2558-001	February 2007	Initial version of document.

About This Publication

The following topics provide more information about this publication:

Purpose

This publication describes the features of the Dialogic® 3G-324M API library and provides programming guidelines for those who choose to develop applications using this API library. It also provides a reference to the functions, events, data structures, and error codes in the Dialogic® 3G-324M API library.

Applicability

This document version is published for Dialogic® Host Media Processing Software Release 4.1LIN.

This document may also be applicable to other Dialogic® software releases (including service updates). Check the Release Guide for your software release to determine whether this document is supported.

Intended Audience

This publication is intended for the following audience:

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

This document assumes that its readers are familiar with the Linux operating system and have experience using the C programming language.

How to Use This Publication

The information in this document is organized in two major parts:

- *Programming Guide* content, which describes the Dialogic® 3G-324M software features, gives background information on the 3GPP 3G-324M technical specification, provides feature implementation guidelines, and discusses debugging utilities.
- *Library Reference* content, which provides a reference to Dialogic® 3G-324M API functions, data structures, events, and error codes.

Related Information

Refer to the following sources for more information:

- For information on the software release, system requirements, release features, and release documentation, see the Release Guide for the software release you are using.
- For details on known issues and late-breaking updates or corrections to the release documentation, see the Release Update for the software release you are using.
- For Dialogic® product documentation, see *http://www.dialogic.com/manuals*
- For Dialogic technical support, see *http://www.dialogic.com/support*
- For Dialogic® product information, see *http://www.dialogic.com*
- For 3GPP Technical Specification 3G TS 26.111, Codec for circuit-switched multimedia telephony service, Modifications to H.324, see *http://www.3gpp.org*
- For ITU-T Recommendation H.324, Terminal for low bit-rate multimedia communication, see *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.324*
- For ITU-T Recommendation H.245, Control protocol for multimedia communication, see *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.245*
- For ITU-T Recommendation H.223, Multiplexing protocol for low bit-rate multimedia communication, see *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.223*

Product Description 1

This chapter describes the Dialogic® 3G-324M API software and provides information about the 3G-324M technical specification. Topics include:

1.1 Product Overview

The Dialogic® 3G-324M API software provides a standards-compliant interface that enables real-time conversational multimedia communication services, specifically video services, to mobile handsets and terminals over circuit-switched networks and packet-switched networks.

The Dialogic® 3G-324M API software is compliant with the 3G-324M technical specification, an umbrella suite of standards produced by the 3rd Generation Partnership Project (3GPP). For background information on the 3G-324M technical specification, see Section 1.2, "3G-324M Technical Specification Overview", on page 15.

The Dialogic® 3G-324M API software provides the following capabilities:

- Ability to control and manage 3G-324M multimedia sessions

  Note: It does not include a call session control protocol such as SS7 ISUP for establishing a bearer channel connection between 3G-324M endpoints.
- Ability to initiate/terminate a 3G-324M session (including H.245 and H.223)
- Ability to interconnect/disconnect H.223 multiplex inputs and outputs (through device management API library functions)

For audio codec and video codec support by platform, see the Release Guide for your software release.

The Dialogic® 3G-324M API can be used in conjunction with other Dialogic® API libraries, such as the Dialogic® Multimedia API library and the Dialogic® Device Management API library, to develop 3G multimedia applications.

1.2 3G-324M Technical Specification Overview

The 3G-324M technical specification is described in the following topics:

- Introduction
- Call Signaling
- H.324 Base Protocol

- H.223 Multiplexer and Demultiplexer Protocol
- H.245 Session Control Protocol
- Media Components
- 3G-324M Session Establishment (Standard)
- Media Oriented Negotiation Acceleration (MONA)

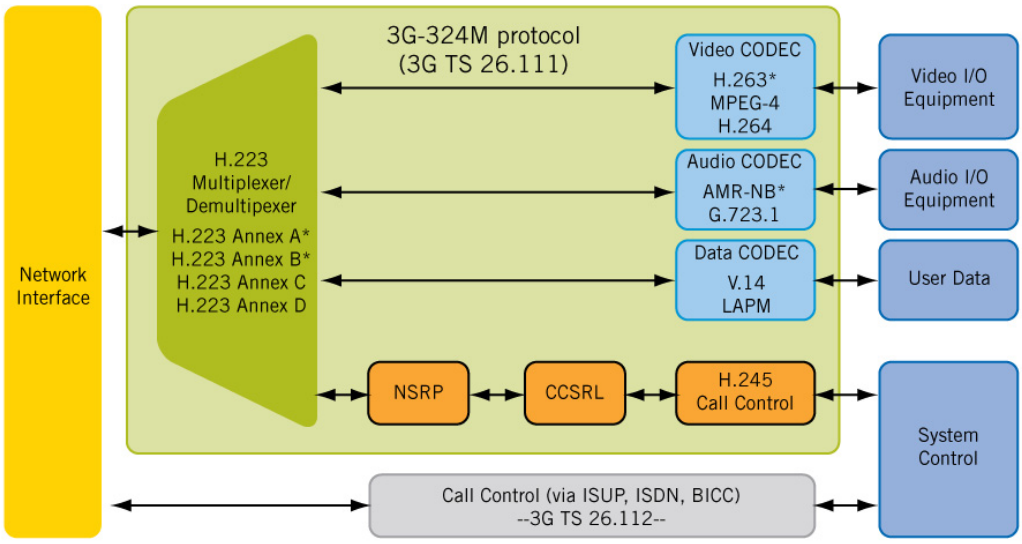
1.2.1 Introduction

The 3G-324M technical specification is an umbrella suite of standards produced by the 3rd Generation Partnership Project (3GPP). It provides a solution for video telephony between 3G-324M endpoints over 3G wireless networks.

An extension to the ITU-T H.324 Recommendation for 3G video telephony, the 3G-324M technical specification includes H.245 for session control; H.223 for bit streams to data packets multiplexer/demultiplexer; H.223 Annex A and B for error handling of low and medium bit error rate (BER) detection, correction, and concealment; and H.324 with Annexes A and C for operating in a wireless environment. H.324 Annex K adds support for Media Oriented Negotiation Acceleration (MONA).

A 3G-324M call involves a network call and a 3G-324M session. A number of different protocols are used to establish a 3G-324M session. A 3G-324M endpoint includes a network interface, a signaling channel, a multiplexer, and media components. The components of the 3G-324M technical specification are shown in Figure 1. An asterisk represents a mandatory component.

Figure 1. 3G-324M Technical Specification



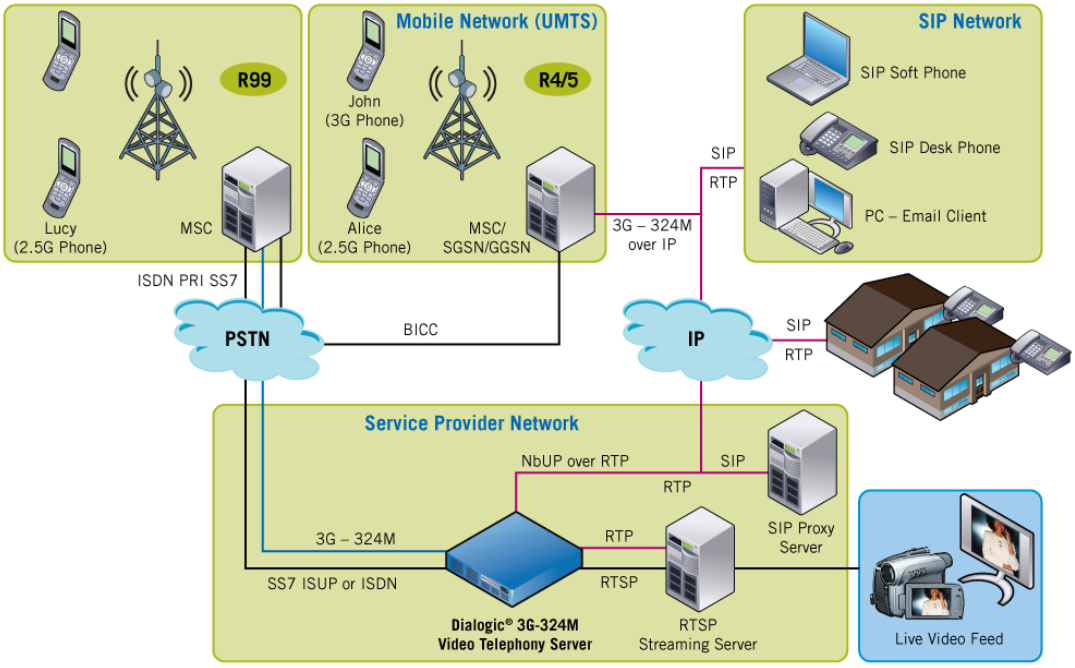
1.2.2 Call Signaling

Call signaling is done in the PSTN network to set up a digital (64 Kbps) bearer channel between two 3G-324M endpoints. Release 99 of the 3G-324M technical specification describes a 3G-324M session in a traditional TDM (PSTN) network. Release 4 and later describe a 3G-324M session in an IP network.

In a TDM network, the call setup of the bearer channel is typically accomplished using the ISUP (SS7) or the ISDN protocol. The digital bearer channel contains no proprietary framing of data, such as A-law or Mu-law companding; it is a clear channel transparent link.

In an IP network, the call is typically established using Bearer Independent Call Control (BICC). The bearer channel in the IP network is an Nb User Plane (Nb UP) RTP connection. Following successful call signaling, the network call is connected, a transparent data bearer channel is established, and the flow of H.223 bitstream data begins the 3G-324M session; see Figure 2.

Figure 2. Call Signaling



1.2.3 H.324 Base Protocol

H.324 is the base protocol for providing real-time multimedia video telephony. It includes the H.223 multiplex protocol and H.245 session control protocol. The H.324 protocol was amended for mobile devices with the H.324M (also known as Annex C) mobile extension. H.324 Annex K adds support for Media Oriented Negotiation Acceleration (MONA).

1.2.4 H.223 Multiplexer and Demultiplexer Protocol

The H.223 protocol is an end-to-end low bitrate protocol in the form of a bitstream between two multimedia endpoints. It provides the method to combine multiple media channels in physical connection with bit-error resiliency and retransmission built in. This is important for wireless applications where transmission errors are prevalent. The H.223 protocol is used for multiplexing and demultiplexing control, audio, and video logical channels of the 3G-324M session into a single bitstream. The bitstream can then be routed over an IP-based or circuit-switched bearer channel.

The H.223 protocol provides two layers: the adaptation layer and the MUX layer.

The adaptation layer provides three different modes of adapting control, audio, and video data for transmission. These three modes are known as AL1, AL2 and AL3, and provide increasing levels of error protection. A separate adaptation layer mode is defined for each of the three data streams prior to multiplexing. The AL1 mode is used for the control channel and adds no additional error protection to the one already provided in the upper layers of the H.245 control channel that it services. The AL2 mode is used for audio and optionally video. The AL3 method may also be used for video in place of AL2, as it provides increased error protection via defined retransmission procedures.

Similarly, the MUX layer provides different modes of framing the multiplexer PDUs (MUX-PDUs). These different modes are called multiplexer levels and provide increased levels of error resilience. The three modes available for use are multiplexer Level 0 (ML0), Level 1(ML1), and Level 2 (ML2).

When the bearer channel is first established, a mobile level detection procedure is started that sets up the frame synchronization between endpoints. Each endpoint sends a frame synchronization flag pattern that corresponds to the highest multiplexer level at which it is capable of operating. At the same time, it detects the incoming flag pattern from its peer. If the synchronization flag pattern received by the endpoint represents a lower multiplex level than it is transmitting, the endpoint changes its transmitted multiplex level to the detected lower level. Each side retransmits its respective synchronization flag patterns until a level is reached that both endpoints support. This protocol helps set the error resiliency level. Once the multiplexer level is determined, the H.245 control channel (also known as Logical Channel 0) is opened automatically between endpoints.

1.2.5 H.245 Session Control Protocol

The H.245 protocol provides session control between 3G-324M endpoints. It provides the method for endpoints to exchange media capabilities, open and close logical channels for media, and specify the content of the media channels when they are opened.

The H.245 protocol begins with Terminal Capabilities Set (TCS) exchange and Master Slave Determination (MSD) exchange.

During TCS exchange, each endpoint specifies its receiver capabilities so that its remote peer may subsequently open logical channels and transmit in a media format and multiplexer format that are supported by both endpoints. The local endpoint (transmitter) opens unidirectional logical channels based on the capabilities provided by the remote endpoint (receiver) in the TCS exchange. Unidirectional logical channels are opened in the forward direction, to specify audio or video

media format from local to remote endpoint. Each endpoint opens one forward logical channel for transmitting audio and one forward logical channel for transmitting video. Logical channels identify the media capabilities which are used to specify codec type and format, and multiplexer capabilities of the particular media channel.

MSD exchange determines that the endpoint with the highest terminal type becomes the master. The endpoint deemed master is given priority in resolving conflicts during logical channel establishment.

1.2.6 Media Components

The 3G-324M technical specification mandates support for the Adaptive Multi-Rate (AMR) codec for audio and the H.263 codec for video. The AMR codec was originally developed for wireless cellular. Dialogic supports AMR-NB and G.723.1 codecs for audio; and H.263 and MPEG-4 codecs for video. Codec support varies by platform; see the Release Guide for your software release for more information.

1.2.7 3G-324M Session Establishment (Standard)

The 3G-324M session requires a few steps to set up a 3G-324M call. The following steps highlight some of the exchanges that occur between endpoints to establish a 3G-324M session. This is also referred to as the "standard" 3G-324M session establishment procedure in this document:

1. A bearer channel is established.
2. A training phase determines the H.223 multiplexing level.
3. Terminal Capabilities Set (TCS) messages and Master Slave Determination (MSD) messages are exchanged.
4. Each endpoint opens video and audio logical channels in the forward direction that include the type of media that is defined for that channel.
5. Multiplex Table Entries are exchanged to define how the logical channel data is organized into multiplexer frames.
6. Media is exchanged between the endpoints.

Note: If Media Oriented Negotiation Acceleration Procedure (MONA) is enabled, a modified set of exchanges occur; see Section 1.2.8.4, "3G-324M Session Establishment with MONA", on page 21.

1.2.8 Media Oriented Negotiation Acceleration (MONA)

The H.324 Annex K Media Oriented Negotiation Acceleration (MONA) standard is described in the following topics:

- Introduction
- Accelerated Connection Procedure (ACP)
- Media Preconfigured Channels (MPC)
- 3G-324M Session Establishment with MONA

1.2.8.1 Introduction

The Media Oriented Negotiation Acceleration (MONA) standard is a group of complementary procedures designed to significantly reduce delay in H.324 call setup time. The procedures include Media Preconfigured Channels (MPC), Accelerated Connect Procedure (ACP), and Signaling Preconfigured Channel (SPC). The MONA standard implemented by Dialogic uses MPC and ACP procedures, which classifies products based on Dialogic® 3G-324M software as Class II MONA terminals per H.324 Amendment K.7.2.1. Dialogic does not support the SPC procedure.

MONA provides a flexible, accelerated channel setup method that depends on an initial exchange of Preference Messages and the execution of a common inference algorithm. MONA also provides faster preconfigured channel setup mechanisms, which do not wait for standard H.245 message acknowledgements, but provide a fallback if the initial media transmission attempts do not succeed.

With MONA, media channels are typically ready for streaming in less than a second, compared with six to eight seconds using the standard H.245 logical channel establishment procedures.

The MONA feature in the Dialogic® 3G-324M software can be selectively controlled per call. If MONA is disabled, the call proceeds using the standard H.245 logical channel establishment procedures. If MONA is enabled, the MONA procedures are attempted. By default, the MONA feature is disabled to maintain backward application compatibility.

To determine if MONA is supported in a Dialogic® software release, see the Release Guide for that software release.

1.2.8.2 Media Preconfigured Channels (MPC)

The Media Preconfigured Channels (MPC) procedure establishes media channels at the earliest possible moment within a 3G-324M session.

Terminals indicate support for MONA MPC by inserting MONA Preference Messages (PM) within special framing flags before beginning the H.223 multiplex level detection procedure.

The MPC call setup procedure allows for media to be established as soon as the last PM is sent. Media can be received on MPC channels as soon as the first PM is received. Results from the PM exchange determine whether receiving and transmitting MPC channels are established and what type of media is supported. It is possible that all media channels or a subset of media channels are established using the MPC procedure. After media is established, MPCs are managed identically to LCs.

For any media channels that are not established as MPCs, the application falls back to the standard H.245 logical channel establishment procedures. The MONA procedure also attempts to use ACP to establish media channels that were not started as MPCs.

1.2.8.3 Accelerated Connection Procedure (ACP)

The Accelerated Connection Procedure (ACP) allows media streaming to begin earlier in the 3G-324M call.

ACP uses H.245 to establish logical channels (LC) in the same way that is used in standard H.245 LC establishment procedure; however, ACP allows media streaming to begin before OLC acknowledgement is received and without the LC's MES transaction being performed. The OLC is initially assumed successful and is processed after media streaming has started. The Multiplex Table Entry used to send or receive media data (on the LCs established using ACP) is embedded within the TCS messages exchanged during the TCS exchange procedure.

From an application point of view, ACP and the standard H.245 logical channel establishment procedures are handled in the same way. Underlying behavioral differences between ACP and standard H.245 procedures are abstracted from the application. The common API behavior is an exchange of TCS, MSD and OLC messages.

1.2.8.4 3G-324M Session Establishment with MONA

The 3G-324M session requires a few steps to set up a 3G-324M call. If MONA is supported in the Dialogic® software and is enabled, the following steps highlight the exchanges that occur between endpoints to establish a 3G-324M session:

1. A bearer channel is established.
2. MONA Preference Messages are exchanged (using MONA synchronization flags).
3. Media is exchanged for all common Media Preconfigured Channels (MPCs) as determined from the MONA Preference Messages.
4. If all desired MPCs were not established, Accelerated Connection Procedure (ACP) is initiated. This means that the Master Slave Determination (MSD) and Terminal Capabilities Set (TCS) messages are exchanged, and logical channels are established as done in the standard procedure. However, as part of the 3G-324M protocol exchange, ACP does not wait for acknowledgement messages (TCSAck, MSDAck) before initiating logical channels and the resulting media. Thus, OLCs and media may be transmitted after the remote peer's TCS is received, and before a TCSAck and an MSDAck are received. Similarly, media may be initiated before receiving an OLCAck.
5. Media is exchanged for any logical channels opened using ACP or standard H.245 OLC.

For information on standard 3G-324M session establishment, see Section 1.2.7, "3G-324M Session Establishment (Standard)", on page 19.

1.3 Dialogic® 3G-324M API Implementation

The Dialogic® 3G-324M API provides the ability to control and manage a collection of 3G-324M endpoints.

The 3G-324M API provides an abstraction for multiplexing and demultiplexing of multimedia between a 3G-324M network and a circuit-switched network or a packet-switched network. It provides an application interface to the H.245 control session and a means to establish audio and video streams. The 3G-324M API enables the multiplexing and demultiplexing of the audio and video streams to/from a bearer channel through its software component, the m3g device.

The Dialogic® 3G-324M software consists of two primary components as follows:

3G-324M (m3g) API library
: Provides software component device abstraction and application interface. It abstracts the H.245 session control, an audio packet interface, and a video packet interface, and exposes them as separate m3g devices:

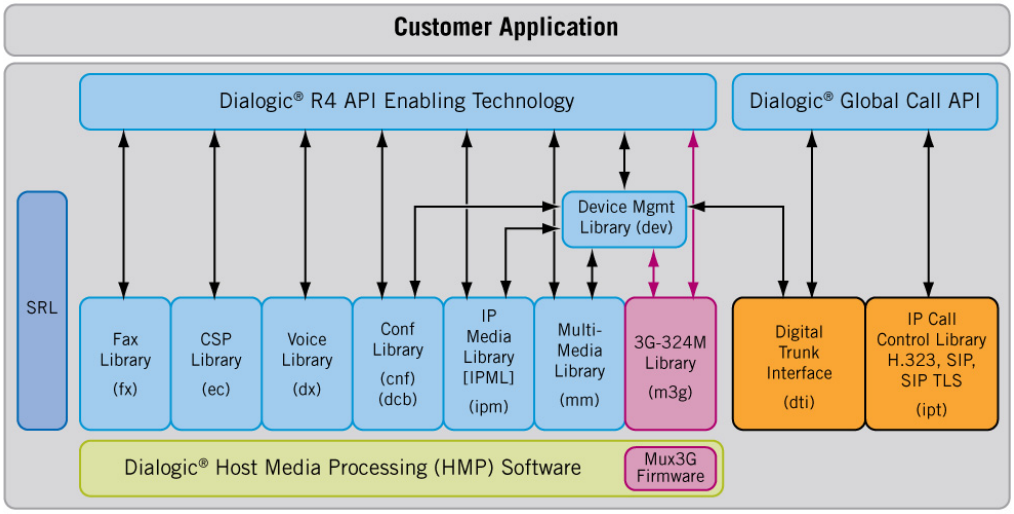
- the m3g control device, used to represent the aggregate interface that connects to the network transport.
- the m3g audio device, used to control the audio stream connection and set the audio media capabilities.
- the m3g video device, used to control the video stream connection and set the video media capabilities.
- the m3g board device, used to set global default values.

For more information on devices, see Chapter 2, "Device Handling".

3G-324M (Mux3G) firmware
: Handles the low-level 3G-324M protocol exchange. The firmware is the 3G session protocol and multiplexing/demultiplexing engine. It controls the 3G-324M protocol stack implementation, and provides user session input and session progress eventing through the 3G-324M API library. It carries out the protocol between the local device and the remote 3G-324M endpoint. It provides the minimum interface necessary for 3G session control, without requiring the application developer to have detailed protocol-specific knowledge.

Figure 3 illustrates the Dialogic® API libraries. The API library support varies by platform. For support information, see the Release Guide for your software release.

Figure 3. Dialogic® API Libraries



Device Handling 2

This chapter describes the devices used in the Dialogic® 3G-324M software. Topics include:

2.1 Device Overview

The Dialogic® 3G-324M software provides device handles to control the 3G-324M endpoint. The device abstractions include board, control, audio, and video devices.

An m3g device represents one instance of a 3G-324M endpoint and terminates the 3G-324M peer-to-peer protocol. The m3g device provides a connection to the aggregate data on the 3G network bearer channel and internal connections for audio and video streams.

When the m3g device receives aggregate data from the 3G network bearer channel, it demultiplexes the data into H.245 control messages, audio streams, and video streams. In the reverse direction, audio and video streams are multiplexed with H.245 control messages and sent on the aggregate 3G network bearer channel to the remote 3G-324M endpoint.

2.2 Device Types

Each 3G-324M endpoint is a composite or aggregate of several device types:

board device

The board device is used to set global default values. The board device handle is used in **m3g_SetParm( )** function calls to specify parameter values for all applicable control, audio, and video device instances subsequently opened on the specified board. It is used in **m3g_EnableEvents( )** function calls to enable unsolicited events. It is also used by other functions such as **m3g_SetVendorId( )** and **m3g_StartTrace( )**.

The board device name is "m3gB*m*", where "*m*" is the specified board number.

Note: Only one board, m3gB1, is currently supported.

Note: Any function called on a board device affects all instances on that board device. For example, if an application resets the board device via **m3g_Reset( )**, all channels are reset on that board device. Similarly, if an application uses the **m3g_SetVendorId( )** and **m3g_StartTrace( )** functions on the board device level, all channels on the board device are affected by these functions. Be aware of this behavior, in particular if two applications access the same board device.

control device

The control device is the primary handle for 3G-324M endpoint control. It is used to manage the following functional interfaces:

- H.245 control – provides H.245 control operations for a given 3G-324M endpoint. This device is automatically associated with the aggregate H.223 multiplex/demultiplex as logical channel 0 when the device is opened.
- H.223 multiplex/demultiplex – permits physical connections to and from the H.223 multiplex/demultiplex over CT Bus timeslots or over IP using an Nb User Plane (Nb UP) protocol. CT Bus timeslots may be used to connect to appropriate T1/E1 bearer channels which transport the aggregate data off-board to route the H.223 multiplex/demultiplex to other 3G-324M endpoints. The Nb UP may be used to route bearer control and transport of the H.223 multiplex/demultiplex within the 3G core network Release 4 and later.

Connections and disconnections between the H.223 multiplex/demultiplex aggregate are made using device management API functions. If the aggregate is routed over a DS0 timeslot on the CT Bus, **dev_Connect( )** and **dev_Disconnect( )** are used. If the aggregate is routed over the Nb UP, **dev_PortConnect( )** and **dev_PortDisconnect( )** are used.

The control device name is "m3gB*m*T*n*", where "*m*" is the specified board number and "*n*" is the specified channel number.

audio device

The audio device represents the audio connection to and from the H.223 multiplex. This device type does not initiate or terminate audio streams. The audio device connects another R4 device type, such as an IP media device (ipmBxCy) or a multimedia device (mmBxCy), which provides the source and destination for the associated audio data streams, through the **dev_PortConnect( )** and **dev_PortDisconnect( )** functions.

Similarly, the audio device may be connected to a digital network interface device (dtiBxTy) or voice device (dxxxBxCy) through the **dev_Connect( )** and **dev_Disconnect( )** functions.

Prior to multiplexing/demultiplexing, each audio device must establish a connection to an R4 audio device using **dev_PortConnect( )**.

The audio device name is "m3gB*m*T*n*:AUDIO*p*", where "m3gB*m*T*n*" is the specified control device into and out of which the audio device should be multiplexed/demultiplexed; "*p*" in "AUDIO*p*" represents the number of the audio instance and is used to differentiate multiple audio devices which may comprise an H.223 aggregate.

Note: Only one audio streaming connection to and from the H.223 aggregate is currently supported.

video device

This device represents the video connection to and from the H.223 multiplex. This device type does not initiate or terminate video streams. The video device connects another R4 device type, such as an IP media device (ipmBxCy) or a multimedia device (mmBxCy), which provides the source and destination for the associated video data streams, through the **dev_PortConnect( )** and **dev_PortDisconnect( )** functions.

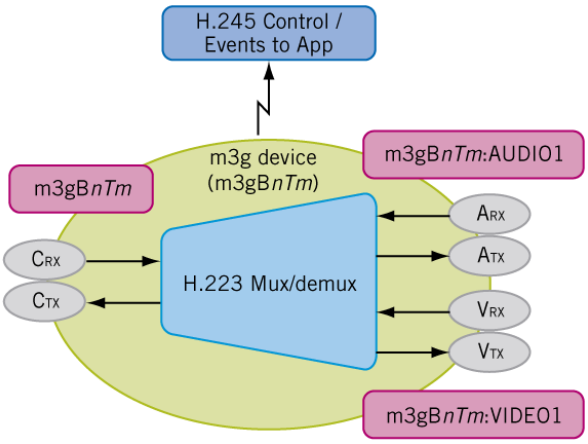
Prior to multiplexing/demultiplexing, each video device must establish a connection to an R4 video device using **dev_PortConnect( )**.

The video device name is "m3gB*m*T*n*:VIDEO*p*", where "m3gB*m*T*n*" is the specified control device into and out of which the video device should be multiplexed/demultiplexed; "*p*" in "VIDEO*p*" represents the number of the video instance and is used to differentiate multiple video devices which may comprise an H.223 aggregate.

Note: Only one video streaming connection to and from the H.223 aggregate is currently supported.

Figure 4 illustrates the 3G-324M devices.

Figure 4. 3G-324M (m3g) Devices



Use the Dialogic® Standard Runtime Library device mapper functions to retrieve information about devices in a system. For more information on device handling, see the *Dialogic® Standard Runtime Library API Programming Guide*.

Event Handling 3

This chapter describes how the Dialogic® 3G-324M software handles events. Topics include:

3.1 Event Handling Overview

Dialogic® 3G-324M events are retrieved using Dialogic® Standard Runtime Library (SRL) event retrieval mechanisms, including event handlers. The SRL is a device-independent library containing event management functions and Standard Attribute functions.

This chapter lists SRL functions that are typically used by 3G-324M applications. For a list of 3G-324M API events, see Chapter 13, "Events".

For more information on event handling, see the *Dialogic® Standard Runtime Library API Programming Guide*.

3.2 Dialogic® Standard Runtime Library Event Management Functions

SRL event management functions retrieve and handle device termination events for library functions. Applications typically use the following functions:

sr_enbhdlr( )
enables event handler

sr_dishdlr( )
disables event handler

sr_getevtdev( )
gets device handle

sr_getevttype( )
gets event type

sr_waitevt( )
waits for next event

sr_waitevtEx( )
waits for events on certain devices

See the *Dialogic® Standard Runtime Library API Library Reference* for function details.

3.3 Dialogic® Standard Runtime Library Standard Attribute Functions

SRL Standard Attribute functions return general device information, such as the device name or the last error that occurred on the device. Applications typically use the following functions:

ATDV_ERRMSGP( )
pointer to string describing the error that occurred during the last function call on the specified device

ATDV_LASTERR( )
error that occurred during the last function call on a specified device. See the function description for possible errors for the function.

ATDV_NAMEP( )
pointer to device name

ATDV_SUBDEVS( )
number of subdevices

See the *Dialogic® Standard Runtime Library API Library Reference* for function details.

Error Handling 4

This chapter describes error handling for the Dialogic® 3G-324M software.

All Dialogic® 3G-324M API functions return a value that indicates the success or failure of the function call. Success is indicated by a return value of M3G_SUCCESS. Failure is indicated by a value of M3G_ERROR.

If a function fails, call the Dialogic® Standard Runtime Library API functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** for the reason for failure. These functions are described in the *Dialogic® Standard Runtime Library API Library Reference*.

For a list of errors, see Chapter 15, “Error Codes”.

Implementing a 3G-324M Session 5

This chapter describes an exemplary scenario for implementing a 3G-324M session using Dialogic® 3G-324M software. Topics include:

5.1 Major Implementation Steps

The major steps to implement a 3G-324M session for multimedia play and record can be described as follows.

Note: These steps assume that you enable Media Oriented Negotiation Acceleration (MONA) in your application for faster media channel setup. For any media channels that are not established as MPCs, the 3G-324M protocol falls back to the standard H.245 logical channel establishment procedures.

1. Establish a 3G-324M session and exchange media:
   a. Initialize Devices
   b. Connect Devices
   c. Establish a Bearer Channel
   d. Establish a 3G-324M Session
   e. MONA ACP and Standard H.245 Logical Channel Establishment
   f. Exchange Media
2. Terminate a 3G-324M session:
   a. Terminate a 3G-324M Session
   b. Disconnect the Bearer Channel
   c. Disconnect Devices

d. Close Devices
e. Exit the application

5.2 Initialize Devices

The following topics describe one way to initialize devices. For information on device types, see Chapter 2, "Device Handling".

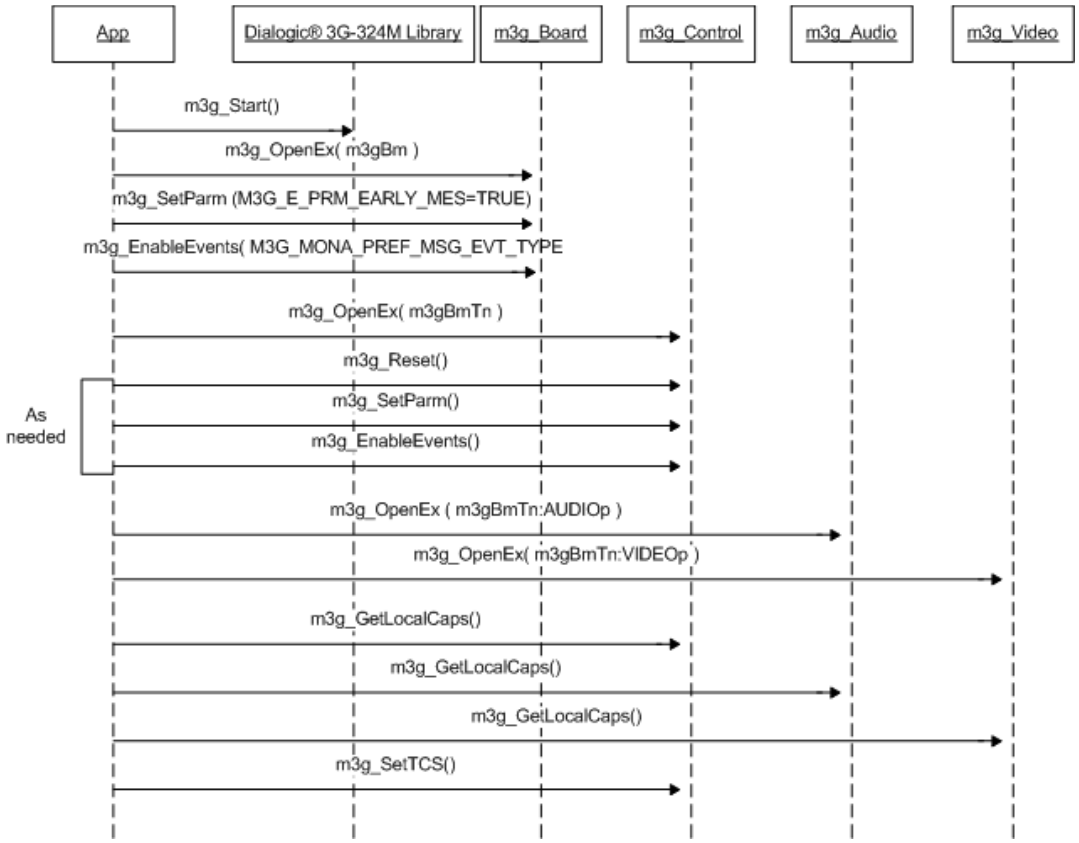
- Initialize the 3G-324M Library
- Open and Configure the m3g Board Device
- Open and Configure the m3g Control Device
- Open and Configure the m3g Audio Device
- Open and Configure the m3g Video Device
- Get Local Capabilities
- Set Preferred Capabilities

5.2.1 Initialize the 3G-324M Library

Call **m3g_Start( )** to initialize the 3G-324M library.

Figure 5 illustrates the steps to initialize devices, starting with 3G-324M library initialization.

Figure 5. 3G-324M Device Initialization



5.2.2 Open and Configure the m3g Board Device

Call **m3g_OpenEx( )** with the board device string to open the board device and initialize default system level parameters for all 3G-324M endpoints in the system.

Call **m3g_SetParm( )** with the board device handle to set system level parameters as needed. Call **m3g_EnableEvents( )** with the board device handle to configure event reporting for optional (unsolicited) events.

Note: Any function called on a board device affects all instances on that board device. For example, if an application resets the board device via **m3g_Reset( )**, all channels are reset on that board device. Similarly, if an application uses the **m3g_SetVendorId( )** and **m3g_StartTrace( )** functions on the board device level, all channels on the board device are affected by these functions. Be aware of this behavior, in particular if two applications access the same board device.

5.2.2.1 Enable MONA Events

To enable delivery of unsolicited MONA events, call **m3g_EnableEvents( )** for a board device or a control device as appropriate and set the M3G_MONA_PREF_MSG_EVT_TYP bitmask. Enabling the following unsolicited events allows the application to be notified when the events are received:

- M3GEV_SEND_MONA_PREF_MSG (MONA preference message sent)
- M3GEV_MONA_PREF_MSG_RCVD (MONA preference message received)

5.2.2.2 Enable Early MES Mode

The "early MES" mode transmits the H.245 MultiplexEntrySend (MES) message without waiting to receive an H.245 OpenLogicalChannelAck (OLCAck) message. As a result, this mode uses one less round trip of H.245 message groupings thereby shortening the time to establish logical channels and media.

To enable early MES mode, use **m3g_SetParm( )** and set M3G_E_PRM_EARLY_MES parameter to true. It is beneficial to enable this mode in your application. Early MES is used during fallback to MONA Accelerated Connection Procedure (ACP) and in standard H.245 logical channel establishment.

5.2.3 Open and Configure the m3g Control Device

For every 3G-324M channel being used by the application, call **m3g_OpenEx( )** with the control device string to open the m3g control device and to configure the H.223 multiplex capabilities.

It is generally good practice to call **m3g_Reset( )** with the m3g control device handle to reset the 3G-324M channel to an initialized state at the beginning of the application. This function is used to make sure that the control, audio, and video devices that may not have been properly closed are returned to an initial state.

Call **m3g_SetParm( )** with the control device handle to set parameters as needed. Call **m3g_EnableEvents( )** with the control device handle to configure event reporting for optional (unsolicited) events.

5.2.4 Open and Configure the m3g Audio Device

For every 3G-324M channel being used by the application, call **m3g_OpenEx( )** to open the m3g audio device to configure the audio stream to and from the H.223 multiplex.

5.2.5 Open and Configure the m3g Video Device

For every 3G-324M channel being used by the application, call **m3g_OpenEx( )** to open the m3g video device to configure the video stream to and from the H.223 multiplex.

5.2.6 Get Local Capabilities

For every 3G-324M channel being used by the application, call **m3g_GetLocalCaps( )** to get the default capabilities supported by the specified device type. Call this function for control, audio, and video device type.

Call **m3g_GetLocalCaps( )** with the m3g control device handle to get the default H.223 multiplex capabilities of the system.

Call **m3g_GetLocalCaps( )** with the m3g audio device handle to get the default audio capabilities of the system, such as audio codec types supported by the system.

Call **m3g_GetLocalCaps( )** with the m3g video device handle to get the default video capabilities of the system, such as video codec types supported by the system.

Default capabilities are returned in the M3G_GET_LOCAL_CAPS_CMPLT termination event.

5.2.7 Set Preferred Capabilities

Set the application's preferred H.223, audio, and video capabilities.

Cache the H.223, audio, and video capability structures returned from **m3g_GetLocalCaps( )** and use these structures to make modifications to the default system capabilities.

Call **m3g_SetTCS( )** with the m3g control device handle to set the local Terminal Capabilities Set (TCS) for H.223, audio, and video. The local TCS can be the entire list of capabilities as received by **m3g_GetLocalCaps( )** or modified values based on the application specification. For interoperability reasons, the H.223 and media capabilities specified should be left unchanged from those values returned from **m3g_GetLocalCaps( )**. Changing the default capability settings may increase the risk of failures in subsequent logical channel establishment. Note that the order and preference of the media capabilities can be changed, but the capabilities themselves should not be changed.

The video receive capability preferences are listed in the array of capabilities in the M3G_CAPS_LIST structure in decreasing order of preference. The most preferred video receive capability is ordered first in the array, while the least preferred is ordered last. This preference is indicated to the remote peer terminal by the ordering of capabilities listed within the AlternativeCapabilitySets in the H.245 TerminalCapabilitySet (TCS) message.

5.3 Connect Devices

After completing the tasks described in Section 5.2, "Initialize Devices", on page 30, proceed to connect devices. The following topics describe how to connect devices:

- Connect Devices Overview
- Connecting Networks
- Connecting Audio and Video Media

- Transcoding Versus Native Connection

5.3.1 Connect Devices Overview

Before multiplexing starts and media is streaming in a 3G-324M session, the m3g device is to be connected on one side to the network aggregate data and on the other side to audio and video media terminations. The m3g device does not terminate the audio and video streams, but performs the multiplexing/demultiplexing, and provides an audio and video connection to other Dialogic® software components, such as the multimedia mm device.

For the m3g device, three full-duplex device connections (network, audio, and video) are to be made to achieve multiplexing and demultiplexing capability. Use the Dialogic® Device Management API library (dev_ functions) to connect devices. For more information, see the *Dialogic®* Device Management API *Library Reference*.

Depending on your application, you may need to open other devices, such as the multimedia (mm) device for multimedia play and record, the IP media streaming (ipm) device for connection to an IP endpoint, or the conferencing (cnf) device for multimedia conferencing.

You decide when to connect and disconnect devices. Devices can be connected statically before call establishment or dynamically once the multimedia services are required. In this section, devices are connected after initialization and remain connected throughout the 3G call. It is up to you to handle dynamic allocation of device resources and the appropriate events to suit your application resource allocation.

5.3.2 Connecting Networks

On the network side, connect the m3g control device to the PSTN or IP-based network device, representing the ingress or egress of the 3G-324M bearer channel. This is the multiplex connection to the aggregate bitstream data provided by the bearer channel.

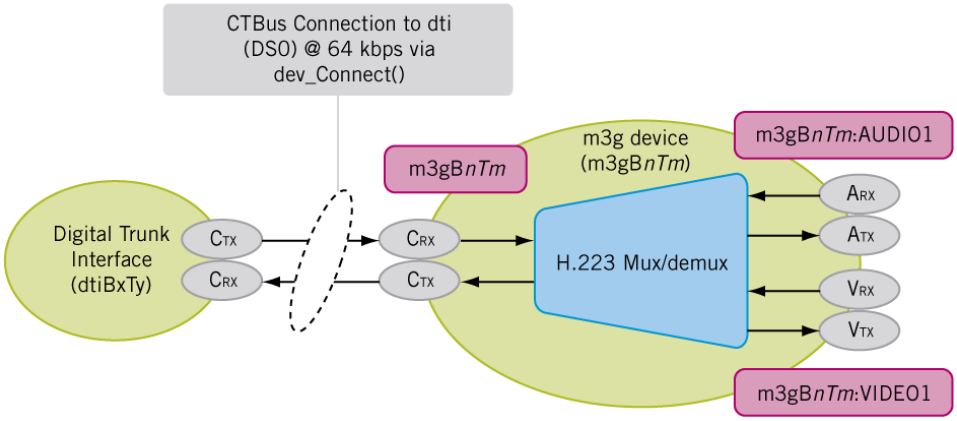
5.3.2.1 PSTN Network

In a PSTN network, connect the m3g control device to a Dialogic® Global Call digital network interface (dti) device using the Dialogic® Global Call API. The dti device provides the PSTN connection, and represents the 64 Kbps transparent clear channel or “unrestricted data” DS0 timeslot connection to the PSTN network. The dti device is set to non-companded, transparent mode to assure that the timeslot carries the unmodified bitstream. Use **dev_Connect( )** to connect the m3g device to a dti device.

Note: In some Dialogic® software releases, transparent mode is enabled via **gc_SetConfigData( )** with CCPARM_TRANSPARENTMODE set to CCDM3FW_TRANSPARENTMODE_ENABLE. For information on transparent mode support, see the Release Guide for your software release.

Figure 6 illustrates the m3g control device to dti device connection.

Figure 6. PSTN Network Connection (m3g to dti)

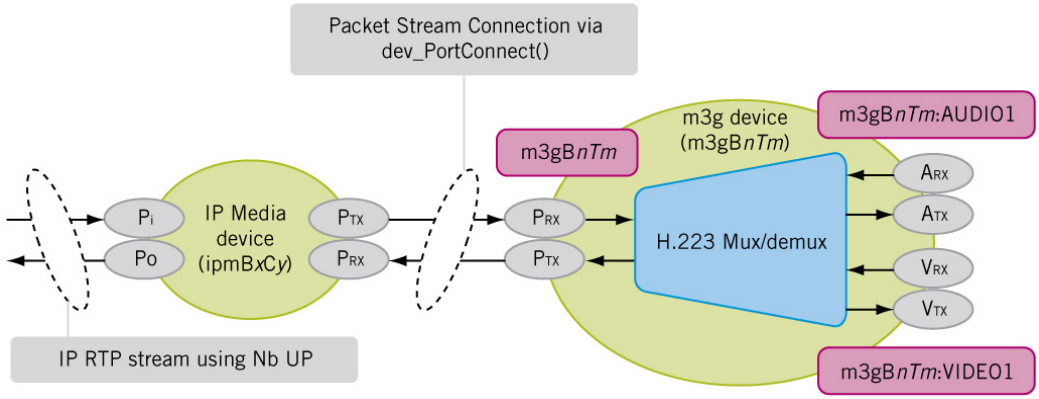


5.3.2.2 IP Network

In an IP network, connect the m3g control device to a Dialogic® IP media streaming device (ipm device). The ipm device is used to control the IP media session over RTP. When used as a transport for 3G-324M bitstream, the ipm device uses the Nb UP protocol and special initialization messages to set up the RTP session. The resulting IP/RTP packets contain an Nb UP packet header and a payload carrying the aggregate bitstream. Use **dev_PortConnect( )** to connect the m3g device to an ipm device.

Figure 7 illustrates the m3g control device to ipm device connection.

Figure 7. IP Network Connection (m3g to ipm)

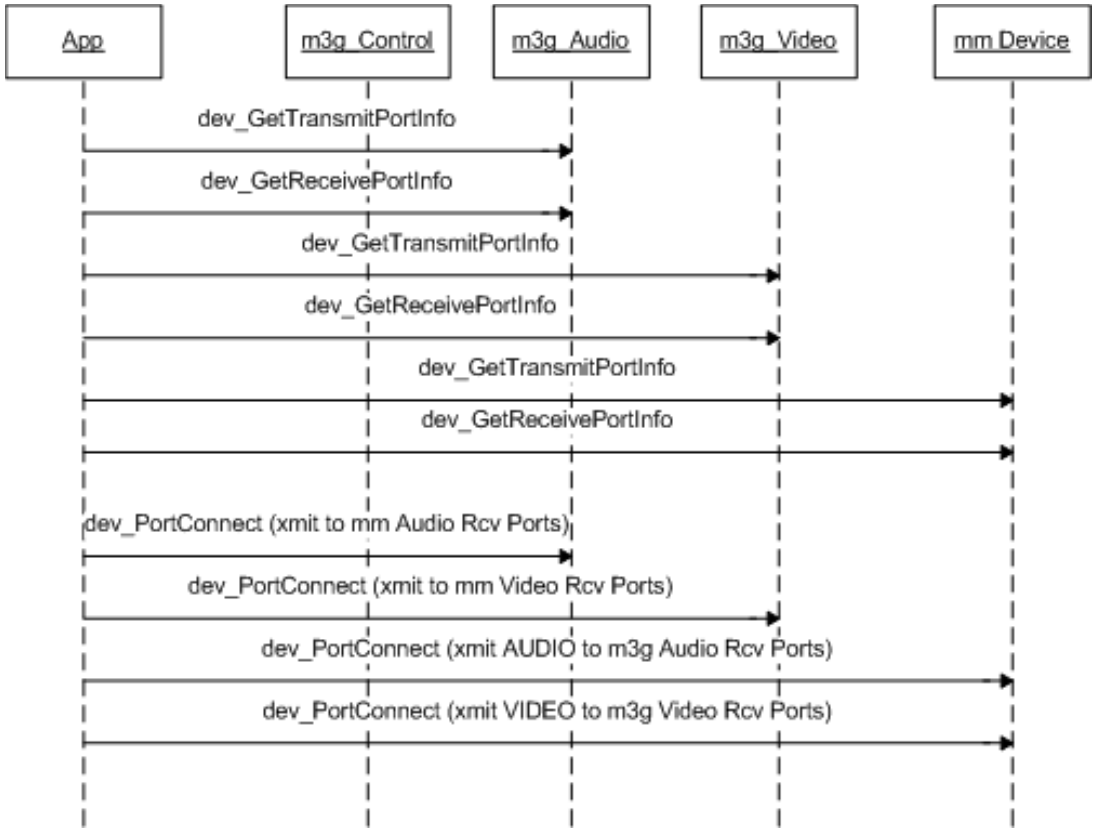


5.3.3 Connecting Audio and Video Media

For a 3G-324M session, media termination connections are made to a multimedia (mm) device, to an IP media streaming (ipm) device, or to a conference (cnf) device using the Dialogic® Device Management API library. To make port connections between devices, get device transmit (Tx) media ports for audio and video, get device receive (Rx) media ports for audio and video, and connect appropriate Tx-Rx pairs between devices using **dev_PortConnect( )**.

Figure 8 provides an overview of how to make media port connections.

Figure 8. Media Connections Overview



Audio from an m3g device can also be connected to voice-only devices, such as the digital trunk interface (dti) device to connect 3G audio to a PSTN audio caller, or to a voice (dx) device for playback/recording of audio files. The audio connections in this case are made using the **dev_Connect( )** function. The video connection will then be unsynchronized and can be made independently to a separate device using the video port connections. For more information, see Section 5.3.3.3, “Connecting Audio to Voice-Only Devices”, on page 38.

5.3.3.1 Retrieving Media Ports

To get the audio and video Tx ports, call **dev_GetTransmitPortInfo( )**. To get the audio and video Rx ports, call **dev_GetReceivePortInfo( )**.

The following steps describe how to get transmit (Tx) and receive (Rx) media ports for the m3g and mm devices (see steps in Figure 9):

a. Call **dev_GetTransmitPortInfo( )** with the m3g audio handle and then with the m3g video handle to get the audio and video Tx ports for the m3g device. The DM_PORT_INFO_LIST of Tx ports for the m3g device are returned in the asynchronous DMEV_GET_TX_PORT_INFO termination event.

b. Call **dev_GetReceivePortInfo( )** with the m3g audio handle and then with the m3g video handle to get the audio and video Rx ports for the m3g device. The DM_PORT_INFO_LIST of Rx ports for the m3g device are returned in the asynchronous DMEV_GET_RX_PORT_INFO termination event.

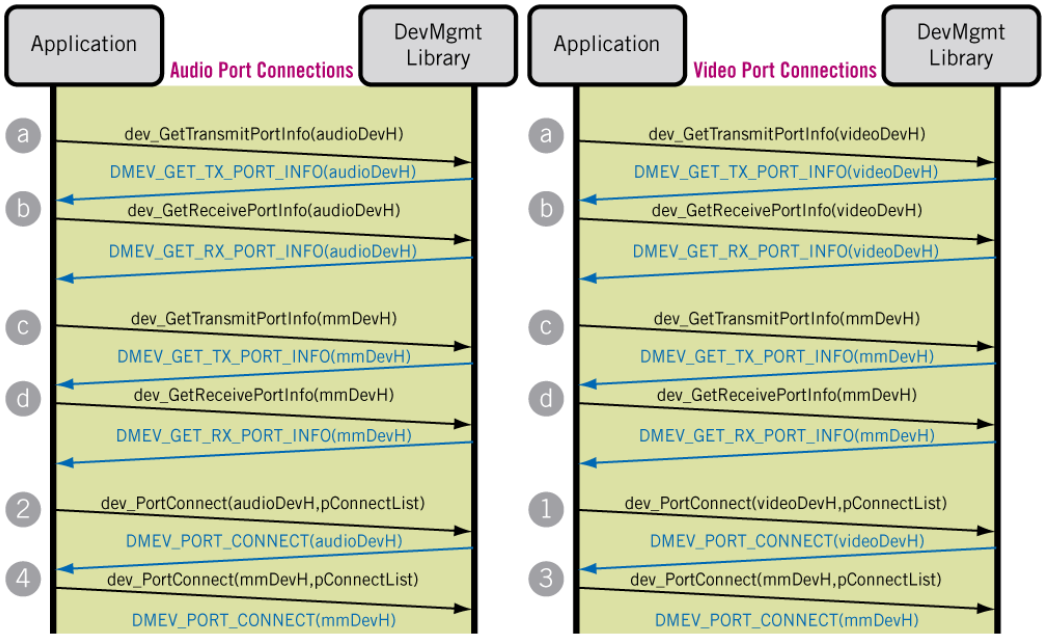
c. Call **dev_GetTransmitPortInfo( )** with the mm handle to get the audio and video Tx ports for the mm device. The DM_PORT_INFO_LIST of Tx ports for the mm device are returned in the asynchronous DMEV_GET_TX_PORT_INFO termination event.

d. Call **dev_GetReceivePortInfo( )** with the mm handle to get the audio and video Rx ports for the mm device. The DM_PORT_INFO_LIST of Rx ports for the mm device are returned in the asynchronous DMEV_GET_RX_PORT_INFO termination event.

e. Cache the audio and video port information for later use when connecting devices.

Figure 9 illustrates audio and video media connections made using the device management API functions.

Figure 9. Retrieving Media Ports and Making Port Connections



5.3.3.2 Connecting Media Ports Between Devices

To make full-duplex audio and video connections between the m3g device and the mm device, call **dev_PortConnect( )**, once for each desired transmit media stream. The **dev_PortConnect( )** function creates a unidirectional packet connection between the transmit port of one device to the receive port of another device.

In the following steps, two audio and two video transmit streams are created, one audio Tx stream and one video Tx stream for each device (see steps in Figure 9):

1. **To connect m3g device Tx video stream**, call **dev_PortConnect( )** with the m3g Tx video port_info list and mm Rx video port_info list.
2. **To connect m3g device Tx audio stream**, call **dev_PortConnect( )** with the m3g Tx audio port_info list and mm Rx audio port_info list.
3. **For multimedia play and record, to connect mm device Tx video stream**, call **dev_PortConnect( )** with the mm Tx video port_info list and m3g Rx video port_info list.
4. **For multimedia play and record, to connect mm device Tx audio stream**, call **dev_PortConnect( )** with the mm Tx audio port_info list and m3g Rx audio port_info list.
5. In each case above, enable transcoding or native (no transcoding) connection by setting the unFlags field of the DM_PORT_CONNECT_INFO data structure. See Section 5.3.4, "Transcoding Versus Native Connection", on page 38.

Note: Tx and Rx directions in these steps represent internal media stream direction relative to the device, not relative to the 3G network.

5.3.3.3 Connecting Audio to Voice-Only Devices

The audio of an m3g device can be connected to a voice-only device, such as the voice (dx) device or the digital trunk interface (dti) device, while the video is connected to a separate device. For example, the audio from a 3G video caller can be connected to a traditional PSTN 2G voice-only endpoint. While the audio is connected between endpoints, the video can be streamed from a separate source, such as an mm device, to the 3G video caller. The audio connection is not synchronized with the video in this case.

To make an audio connection between the m3g device and a voice-only device, call the **dev_Connect( )** function with the "m3g:BnTmAUDIO" device handle and the voice-only device handle. The **dev_Connect( )** function creates a full-duplex or half-duplex audio connection between the two devices based on the full-duplex or half-duplex connection flags. To complete a multimedia connection to the m3g device, connect the video ports to a separate device using the **dev_PortConnect( )** function as discussed in Section 5.3.3.2, "Connecting Media Ports Between Devices", on page 38.

5.3.4 Transcoding Versus Native Connection

You can make a connection to an m3g device in one of two modes: native (no transcoding) or transcoding enabled.

5.3.4.1 Native Connection

In a native connection, audio or video media is passed without modification from the 3G network to the multimedia file. No audio or video transcoding is performed on the media from the 3G network. The media is stored in a multimedia file using the 3G network codecs. In most cases, the video codec is stored as H.263 or MPEG-4 for video and as AMR-NB for audio. Consider selecting native connection mode to save on MIPs and to increase density.

To make a native unidirectional media stream connection, call **dev_PortConnect( )** with unFlags set to DMFL_TRANSCODE_NATIVE in the DM_PORT_CONNECT_INFO structure.

5.3.4.2 Transcoding Connection

Transcoding can be enabled to keep multimedia files in a singular format. Audio transcoding allows the network coder, for example AMR-NB, to be stored as PCM data. Video transcoding is a CPU-intensive process that provides translation between video coder types, such as H.263 and MPEG-4, resolution types, bit rates, and frame rates. Enabling video transcoding also provides greater control over video I-frame generation and stream manipulation, such as text overlay; however, the greater control comes at the expense of CPU and reduced density.

Transcoding support varies by platform or software release. For support information, see the product-specific Release Guide.

To enable transcoding on a unidirectional media stream, call **dev_PortConnect( )** with unFlags set to DMFL_TRANSCODE_ON in the DM_PORT_CONNECT_INFO structure. Transcoding takes place between the two media coder formats specified in each device's data structures.

For transport over 3G, the media coder settings are specified for the m3g device in the **m3g_SetTCS( )** and **m3g_OpenLC( )** data structure initialization.

For multimedia play and record, the media coder settings are specified for the mm device in the **mm_Play( )** and **mm_Record( )** data structure initialization. Transcoding is performed between devices by translating the media stream definitions at the m3g device to and from the media definitions at the mm device.

5.4 Establish a Bearer Channel

After completing the tasks described in Section 5.3, "Connect Devices", on page 33, your application is ready to receive a 3G call from the network or to make a 3G call to the network and establish a bearer channel. The 3G call is established and connected using signaling means outside of the 3G-324M protocol and the 3G-324M API library.

In a PSTN network, the bearer channel is typically established using ISUP or ISDN protocol. The bearer channel in this network is a 64 Kbps clear channel, "unrestricted data" DS0 timeslot.

In an IP network, the call is typically established using BICC. The bearer channel in this network is an Nb UP RTP connection.

5.5 Establish a 3G-324M Session

After a call is connected within the network and a bearer channel is established as described in Section 5.4, "Establish a Bearer Channel", on page 39, the 3G-324M protocol is started with the remote 3G-324M endpoint. The role of the 3G-324M API begins after the application receives indication that a bearer channel is established. At this time a 3G-324M session can be established through 3G-324M protocol exchange.

These instructions assume that you have enabled Media Oriented Negotiation Acceleration (MONA) in your application for faster media channel setup. For more information on MONA, see Section 1.2.8, "Media Oriented Negotiation Acceleration (MONA)", on page 19.

The following topics provide information about establishing a 3G-324M session with MONA enabled:

- Start H.245 with MONA
- Establish MONA MPCs
- Exchange Media Using MONA

5.5.1 Start H.245 with MONA

To start a 3G-324M session with MONA, call **m3g_StartH245( )** on the m3g control device handle with MONA enabled to initialize the H.223 multiplex/demultiplex. MONA is enabled in the M3G_H223_SESSION structure.

Before H.223 multiplex level framing, a MONA preference message is exchanged within the frame header to identify the local Tx and Rx media preconfigured channels (MPC) supported by each endpoint. The following unsolicited events are returned to indicate that the MONA preference messages have been exchanged by the local and remote endpoints:

- M3GEV_SEND_MONA_PREF_MSG
- M3GEV_MONA_PREF_MSG_RCVD

Data about supported MPCs is delivered with each event. For more information about this data, see the M3G_MONA_TXRX_MPC_SUPPORT data structure.

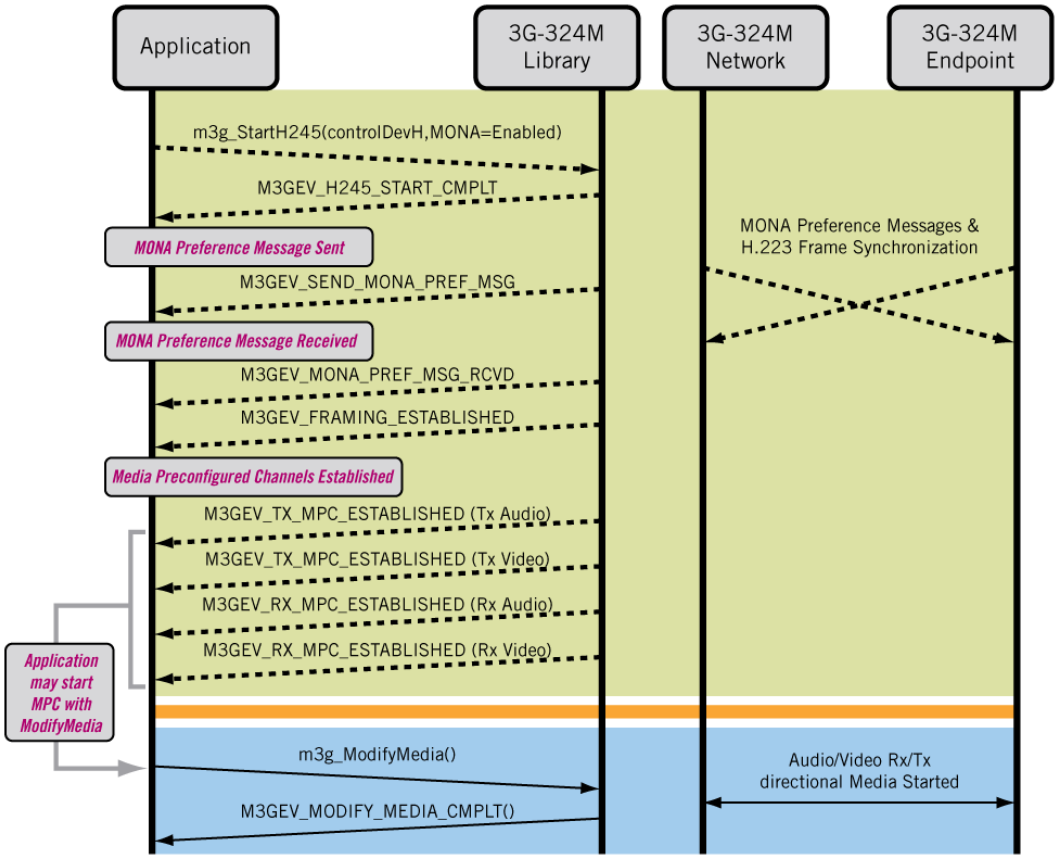
After **m3g_StartH245( )** completes successfully, the following event is returned indicating that the framing layer is sufficient to establish the H.223 abstraction layer:

- M3GEV_FRAMING_ESTABLISHED

In MONA procedures, the MPC-TX and MPC-RX bits are automatically set based on the capabilities specified in **m3g_SetTCS( )**. If the specified terminal capabilities match the format of the MPCs as defined in the MONA standard, the respective MPC-TX bits are automatically set. However, only one MPC-RX video bit may be set for a given call, with the priority given to H.263. Thus, if the default capabilities are unchanged in the call to **m3g_SetTCS( )**, the MPC-TX bits for AMR, H.263, and MPEG-4 are enabled, and the MPC-RX bits for AMR and H.263 are enabled.

Figure 10 illustrates the bearer channel establishment and media exchange with MONA.

Figure 10. 3G-324M Session with MONA Sequence



5.5.2 Establish MONA MPCs

Up to four MONA media preconfigured channels (MPCs) can be established based on the MONA preference messages: audio Tx, audio Rx, video Tx, and video Rx.

After an exchange of MONA preference messages, the following events are returned to indicate that the MPCs have been established and that media streaming can be started:

- M3GEV_TX_MPC_ESTABLISHED
- M3GEV_RX_MPC_ESTABLISHED

Data about the outgoing or incoming MPCs is delivered with each event. For more information about this data, see the M3G_MONA_MPC data structure.

Each of these events can be returned twice, once for audio and once for video. Note that the application may not receive four events if the MPC negotiation is unsuccessful for any media channel direction. Whether the 3G endpoint does not support MONA or MONA preferences are incompatible, this is a normal situation and the fallback scenario is for the media channels to be opened using the standard H.245 logical channel establishment procedure.

5.5.3 Exchange Media Using MONA

After the MPCs are established, call **m3g_ModifyMedia( )** to start media streaming for each media channel direction that was established. For details, see the Modify Media approach in Section 5.7.2, "Exchange Media Using m3g_ModifyMedia( )", on page 47.

See Figure 10 for an illustration of the messages exchanged using the MONA procedure.

If fewer than four MPCs are established, the ACP and standard H.245 logical channel establishment procedures are used to establish the remaining media channels; see Section 5.6, "MONA ACP and Standard H.245 Logical Channel Establishment", on page 42.

5.6 MONA ACP and Standard H.245 Logical Channel Establishment

After the application receives the M3GEV_FRAMING_ESTABLISHED, M3GEV_MSD_ESTABLISHED, M3GEV_REMOTE_TCS_RCVD, and M3GEV_LOCAL_TCS_ACKD events, the application will not receive any further M3GEV_TX_MPC_ESTABLISHED or M3GEV_RX_MPC_ESTABLISHED events.

Any media channels that were not established as MONA MPCs must be established by getting matched capabilities and by using the standard H.245 logical channel procedure for forward or reverse logical channel establishment. As part of the MONA procedures, when MONA is enabled during **m3g_StartH245( )**, the MONA Accelerated Connection Procedure (ACP) will be attempted as part of the remaining protocol exchange. From the application point of view, the procedure to establish logical channels using ACP is the same as the standard H.245 logical channel procedure described in this document.

Write your application to follow the standard H.245 open logical channel establishment procedure whenever a media channel is not established as an MPC. If MONA is enabled during **m3g_StartH245( )**, any media channels that are not established using the MPC procedure will be established using ACP following these steps. Similarly, if MONA is not enabled during **m3g_StartH245( )**, then all media channels will be established as logical channels using these steps.

The following topics describe how to establish a 3G-324M session using MONA ACP and standard H.245 logical channel procedure:

- Start H.245 – Standard Open Logical Channel Procedure
- Get Matched Capabilities
- Open Audio/Video Logical Channels (OLC)

5.6.1 Start H.245 – Standard Open Logical Channel Procedure

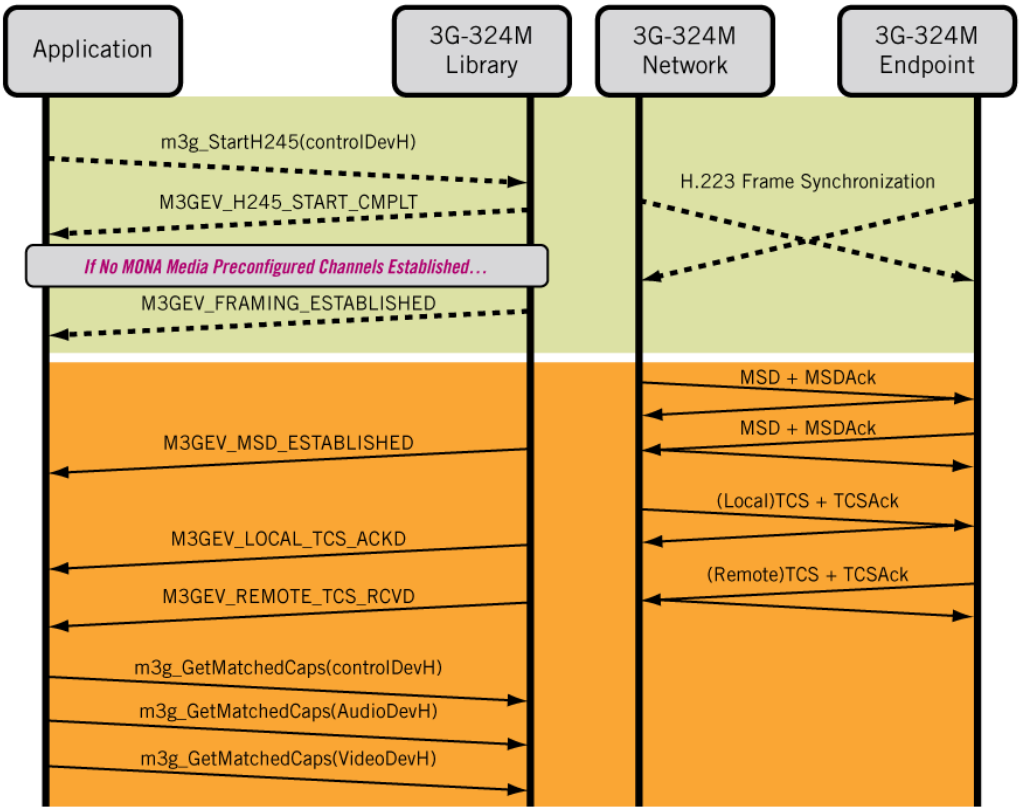
After MONA Preference Messages are exchanged, the 3G-324M protocol continues by synchronizing to the H.223 multiplex level framing.

This training phase at the H.223 level determines the error protection capabilities and multiplexing level using the M3G_H223_CAPABILITY structure provided in **m3g_SetTCS( )**.

After the firmware establishes framing, the physical layer is established and the M3GEVT_FRAMING_ESTABLISHED event is returned. The firmware then opens logical channel 0 for H.245 messaging. The initial H.245 exchange, including Master Slave Determination (MSD) and Terminal Capabilities Set (TCS) Exchange, is negotiated with values provided during device initialization, described in Section 5.2, "Initialize Devices", on page 30.

Figure 11 illustrates the sequence for standard 3G-324M session establishment.

Figure 11. 3G-324M Session (Standard) Sequence



5.6.1.1 Terminal Capabilities Set (TCS) Exchange

The TCS transactions are exchanged with the remote 3G-324M endpoint based on the settings provided earlier in **m3g_SetTCS( )**, as described in Section 5.2.7, "Set Preferred Capabilities", on page 33. Local TCS acknowledgement and receipt of remote TCS events are posted by the firmware. The M3GEV_REMOTE_TCS_RCVD and the M3GEV_LOCAL_TCS_ACKD events indicate successful exchange of TCS transactions for each endpoint.

5.6.1.2 Master Slave Determination (MSD) Exchange

The Master Slave Determination transactions are exchanged with the remote 3G-324M endpoint using the terminal type specified in M3G_E_PRM_H245_TERMINAL_TYPE parameter setting (M3G_PARM_INFO structure). When MSD negotiation is complete, the M3GEV_MSD_ESTABLISHED event is returned to the application indicating master or slave designation.

5.6.1.3 Required Events

The application should receive the following events before opening audio and video logical channels:

- M3GEV_FRAMING_ESTABLISHED
- M3GEV_MSD_ESTABLISHED
- M3GEV_REMOTE_TCS_RCVD
- M3GEV_LOCAL_TCS_ACKD

5.6.1.4 Loss of Frame Synchronization

If the frame synchronization pattern is not found in the bitstream, the M3GEV_FRAMING_LOST event is returned.

This condition can occur if the frame synchronization pattern is not detected within 5 seconds of calling **m3g_StartH245( )** after call connection. Additionally, this event is returned if the call is lost or if the remote endpoint drops the 3G-324M session without tearing down logical channels.

If the M3GEV_FRAMING_LOST event is returned within 5 seconds of calling **m3g_StartH245( )**, this may indicate that (1) the data on the aggregate is not properly 3G-324M encoded data, (2) the **m3g_StartH245( )** function was called too early or too late and has timed out, or (3) the data is not present in the bearer channel because the specified timeslot is incorrect.

5.6.2 Get Matched Capabilities

Before opening audio and video logical channels, call **m3g_GetMatchedCaps( )** to determine the matching capabilities between the local and remote 3G-324M endpoints. Call this function after the events described in Section 5.6.1.3, “Required Events”, on page 44 have been received.

The **m3g_GetMatchedCaps( )** function is provided as a convenience to determine the matched capabilities and priority of capabilities between the remote and local 3G-324M endpoints. The capabilities are listed in decreasing order of preference by the endpoint deemed the master in the H.245 Master Slave Determination.

You can use the matched capabilities provided or modify these capabilities to select which capabilities to use to open the forward logical channels for audio and video.

Note: The media or multiplexer format chosen by the local endpoint must be within the tolerances of media formats supported by the remote endpoint as specified in the matched capability list. For example, the local transmitter should not open a video logical channel with a video maxBitRate

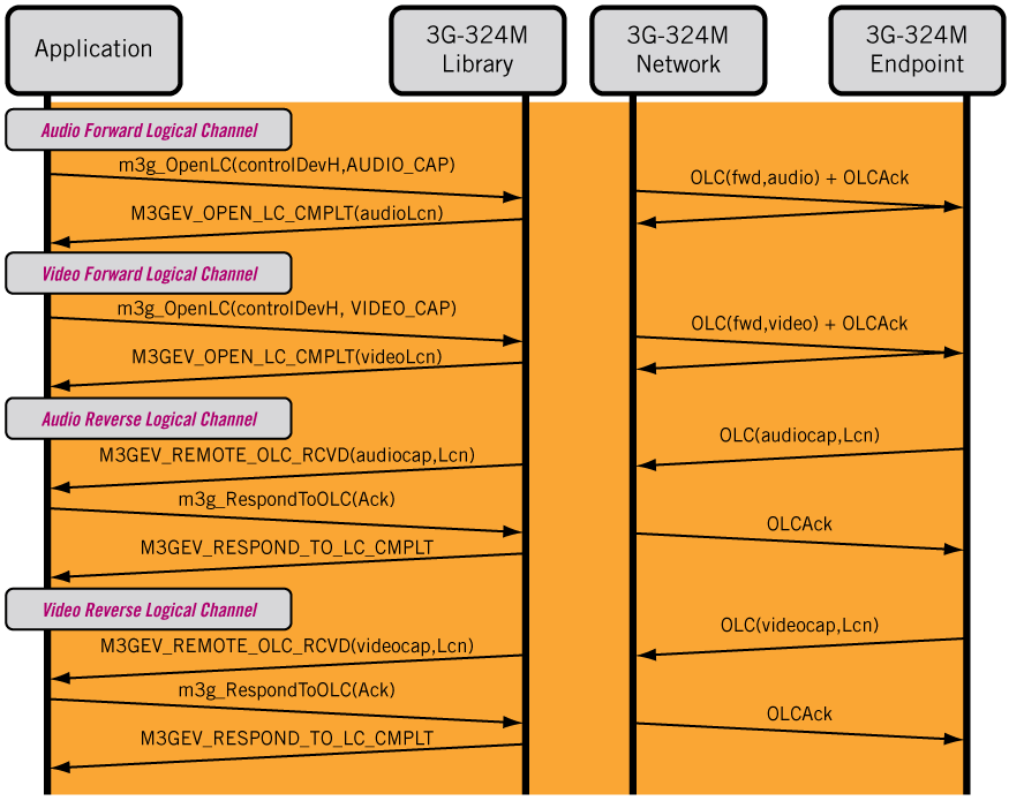
value exceeding the bit rate value indicated by the remote video receiver. To improve interoperability among third-party 3G vendors, choose audio and video codecs that are most preferred by the master. These codecs are identified as the first transmit audio and video codecs in the matched capability list.

5.6.3 Open Audio/Video Logical Channels (OLC)

After the low-level protocol exchange has succeeded and your application determines the matched capabilities between 3G-324M endpoints as described in Section 5.6.1, "Start H.245 – Standard Open Logical Channel Procedure", on page 42, the application can begin opening the forward logical channels for audio and video. The application also acknowledges or rejects the reverse open logical channel requests from the remote endpoint.

Figure 12 illustrates the open logical channel sequence.

Figure 12. Open Logical Channel Sequence



5.6.3.1 Forward Logical Channels

Open Logical Channel (OLC) requests are sent from the local 3G-324M endpoint to the remote 3G-324M endpoint to specify the H.223 adaptation layer format and the media codec to be used in

the transmission of a given media channel. The remote endpoint acknowledges the OLC with an OLCAck. By acknowledging the OLC request, the remote endpoint verifies that the OLC request conforms to its negotiated terminal capabilities and that it will configure its multiplexer and media codec to receive the media data with the specified parameters.

Call **m3g_OpenLC( )** with the control handle and **capabilityType** set to M3G_E_AUDIO_CAPABILITY to open the audio forward logical channel. Call this function with **capabilityType** set to M3G_E_VIDEO_CAPABILITY to open the video forward logical channel.

The OLC is used to specify the media stream transmitted from the local endpoint to the remote endpoint. The media capability, such as audio or video codec type of the channel, is specified by the **pMediaCapability** parameter; the adaptation layer format of the channel is specified by the **pH223LCParameter** parameter. The logical channel number is returned in the OpenLogicalChannelAck received from the remote 3G endpoint, indicated by the M3GEV_OPEN_LC_CMPLT event.

5.6.3.2 Reverse Logical Channels

The Open Logical Channel (OLC) requests from a remote 3G-324M endpoint specify the format of the media channels streaming from the remote endpoint to the local endpoint. A reverse logical channel request (audio or video) is acknowledged or rejected by the application.

Similarly to the local 3G-324M endpoint, the remote 3G-324M endpoint asynchronously requests to open its logical channels specifying the adaptation layer and media codec to use for its media channels. Any OLC received from the remote endpoint specifies the media channel in the direction from the remote endpoint to the local endpoint. The application receives an M3GEV_REMOTE_OLC_RCVD event upon receiving an OLC request from the remote endpoint that conforms to its specified receive media capabilities. It must then use **m3g_RespondToOLC( )** to acknowledge or reject the request; an OLCAck or OLCReject message is sent to the remote 3G-324M endpoint.

After the logical channel has been successfully opened, media streaming can begin in that direction.

5.7 Exchange Media

After logical channels are opened as described in Section 5.6.3, “Open Audio/Video Logical Channels (OLC)”, on page 45, media can flow in that direction. Two methods are available to start media streams between the multimedia device and the H.223 multiplex/demultiplex.

The first method uses **m3g_StartMedia( )** to start a full-duplex media stream after both the forward and reverse logical channels are opened between the local and remote endpoints.

The second method uses **m3g_ModifyMedia( )** to start a unidirectional media stream, which corresponds to the direction of the individual forward or reverse logical channel that was successfully opened.

To simplify the start of bidirectional media streaming to a remote 3G endpoint, consider using the **m3g_StartMedia( )** approach. Conversely, you can achieve immediate unidirectional media in the specified direction if media streams are started using the **m3g_ModifyMedia( )** approach after successful logical channel establishment.

The following topics describe how to exchange media:

- Exchange Media Using m3g_StartMedia( )
- Exchange Media Using m3g_ModifyMedia( )
- Start Multimedia Play and Record
- H.245 UII Digit Detection/Generation
- Video Fast Update Request Detection/Generation

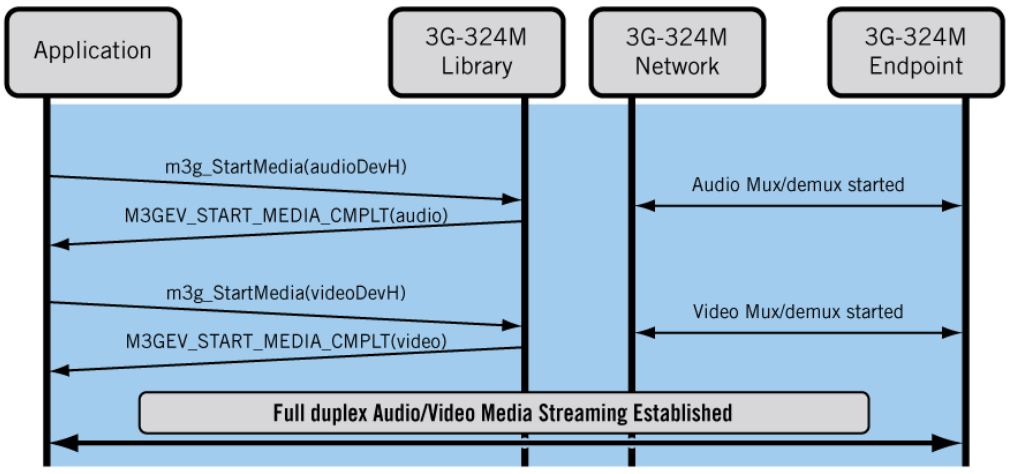
5.7.1 Exchange Media Using m3g_StartMedia( )

The **m3g_StartMedia( )** function begins the H.223 multiplexing/demultiplexing of audio or video data to enable media streaming to the 3G-324M network.

Call **m3g_StartMedia( )** after both the forward and reverse logical channels are open for the media type. Specify the m3g audio handle to start audio streaming. Specify the m3g video handle to start video streaming. The system returns the M3GEV_START_MEDIA_CMPLT event after streaming is successfully initiated.

Figure 13 illustrates the sequence when using **m3g_StartMedia( )**.

Figure 13. Exchange Media – m3g_StartMedia( ) Sequence



5.7.2 Exchange Media Using m3g_ModifyMedia( )

The **m3g_ModifyMedia( )** function begins the H.223 multiplexing/demultiplexing of audio or video data in the specified half-duplex direction to the 3G-324M network.

Call **m3g_ModifyMedia( )** with the appropriate audio or video handle after the corresponding forward logical channel or reverse logical channel is open. Indicate the direction of the media stream, such as transmit only (M3G_E_TX), receive only (M3G_E_RX), transmit and receive (M3G_E_RXTX), or complete stop (M3G_E_IDLE). This direction is relative to the 3G network. This is opposite to the direction specified for port connections in Section 5.3.3.2, "Connecting Media Ports Between Devices", on page 38, which was relative to the media termination device.

Example Scenario

The following example shows one way to start unidirectional media streaming corresponding to the OLC direction, after the logical channel is open. The sequence in which logical channels may be opened is non-deterministic. Each audio and video logical channel can be opened in parallel by both endpoints in any order.

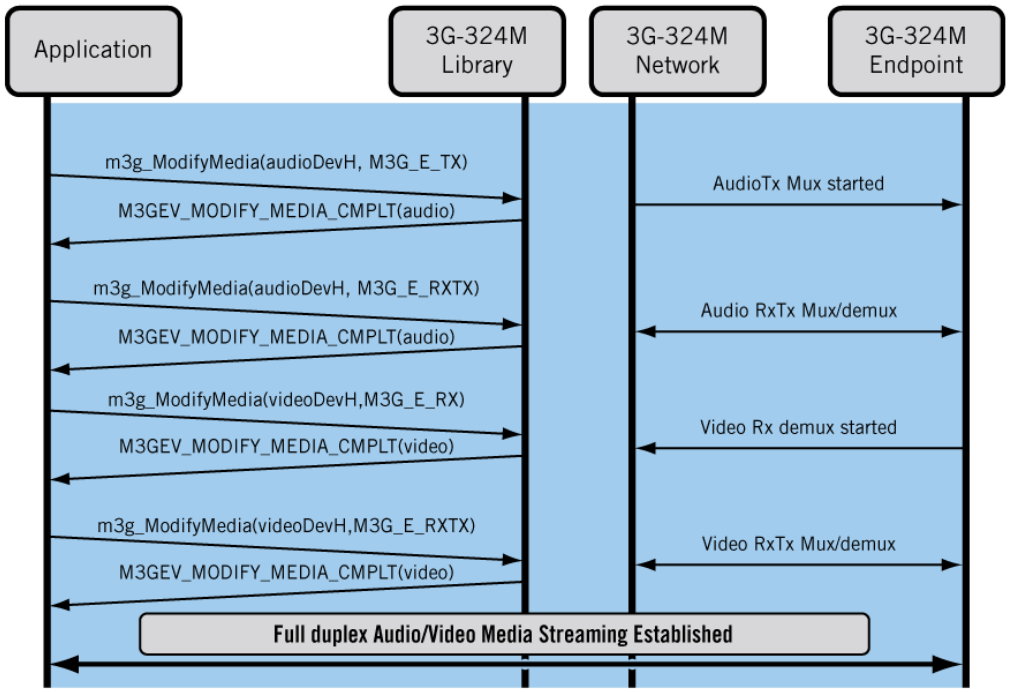
- **Start forward logical channel audio Tx.** Call **m3g_ModifyMedia( )** with the m3g audio handle and the direction M3G_E_TX.
- **Start reverse logical channel audio Rx.** Call **m3g_ModifyMedia( )** with the m3g audio handle and the direction M3G_E_RXTX to enable full-duplex audio.
- **Start reverse logical channel video Rx.** Call **m3g_ModifyMedia( )** with the m3g video handle and the direction M3G_E_RX.
- **Start forward logical channel video Tx.** Call **m3g_ModifyMedia( )** with the m3g video handle and the direction M3G_E_RXTX to enable full-duplex video.

  Note: Tx and Rx directions above represent the media stream direction relative to the 3G network.

The system returns the M3GEV_MODIFY_MEDIA_CMPLT event when streaming is successfully initiated.

Figure 14 illustrates the sequence when using **m3g_ModifyMedia( )**.

Figure 14. Exchange Media – m3g_ModifyMedia( ) Sequence



5.7.3 Start Multimedia Play and Record

After the media streams are started between the H.223 multiplex/demultiplex and the multimedia device, the mm device can be used to play or record multimedia files. The method of connection between devices, native or transcoding, determines the media format that can be used to store media files. For more on native and transcoding, see Section 5.3.4.1, "Native Connection", on page 39 and Section 5.3.4.2, "Transcoding Connection", on page 39, respectively.

For a native connection, the multimedia file must be stored in AMR-NB or G.723 for audio, and H.263 QCIF for video with a video bitrate less than 40 Kbps. For platforms that support MPEG-4, the video can also be stored as native MPEG-4 data.

For a transcoding connection, the multimedia file can be stored in a supported format and it will be transcoded to the proper format. When transcoding is enabled, the media characteristics for the 3G network are specified in the **m3g_OpenLC( )** initialization structures. The media characteristics of the multimedia storage format are specified in the **mm_Play( )** or **mm_Record( )** data structure initialization.

5.7.4 H.245 UII Digit Detection/Generation

Sending DTMF digits over the 3G network is typically accomplished by use of H.245 UserInputIndication (UII) messages. To initiate an H.245 UII message to the remote 3G terminal, call **m3g_SendH245UII( )** specifying the digit mask to be sent.

After receiving an incoming H.245 UII message from the remote terminal, an M3GEV_H245_UII_RCVD event is queued containing the received digit mask. (See **m3g_EnableEvents( )** for more information on enabling events.)

Out-of-band DTMF digits received via UII messages may also be automatically forwarded to the attached Dialogic® audio devices without application intervention. To do so, use **m3g_SetParm( )** with the M3G_E_PRM_RELAY_DIGIT_TO_MEDIA_DEV parameter set to true.

Similarly, when streaming to the 3G network, any RFC 2833 digits detected in the audio stream from the Dialogic® audio devices may be automatically converted to H.245 UII messages. To do so, use **m3g_SetParm( )** with the M3G_E_PRM_RELAY_DIGIT_TO_H245UII parameter set to true.

5.7.5 Video Fast Update Request Detection/Generation

In a 3G-324M session, an endpoint may need to request a full video frame update, called an I-Frame update, from the remote endpoint to provide a reference video frame to facilitate decoding or to refresh the video image. The endpoint requests an I-Frame from the remote terminal by sending an H.245 Miscellaneous Command Video Fast Update (VFU) message.

To send an H.245 VFU request to a remote endpoint, call **m3g_SendH245MiscCmd( )** with the m3g control device handle and set **h245MiscCmdType** to M3G_E_FAST_UPDATE_PICTURE. Requesting a VFU is helpful at the start of a recording, because the **mm_Record( )** function waits up to 5 seconds for an I-frame to start the recording to assure that the beginning of the video is not corrupt.

In the opposite direction, a remote endpoint can send the H.245 VFU request to the application. The M3GEV_H245_MISC_CMD_RCVD event is generated to indicate that the H.245 VFU is received. To automatically generate an I-frame upon VFU request from the remote endpoint, use the **m3g_SetParm( )** function with M3G_E_PRM_RELAY_FASTUPDATE_TO_MEDIA_DEV set to true. Automatic I-frame generation is only valid when a video transcoding connection is made between the m3g device and the mm device during **dev_PortConnect( )**.

5.8 Terminate a 3G-324M Session

Sometime after media streaming has been established, the 3G-324M session can be terminated through the 3G-324M protocol. The following topics describe how to tear down a 3G-324M call and terminate the 3G-324M session:

- Stop Media Streaming
- Terminate the H.245 Session

5.8.1 Stop Media Streaming

Call **m3g_StopMedia( )** with the m3g audio handle to stop audio streaming to and from the H.223 multiplex/demultiplex. Call this function with the m3g video handle to stop video streaming to and

from the H.223 multiplex/demultiplex. The system returns the M3GEV_STOP_MEDIA_CMPLT event when media streaming termination is complete.

5.8.2 Terminate the H.245 Session

After stopping audio and video media streaming in both directions as described in Section 5.8.1, "Stop Media Streaming", on page 50, the final step in tearing down the 3G-324M session is to terminate the H.245 session.

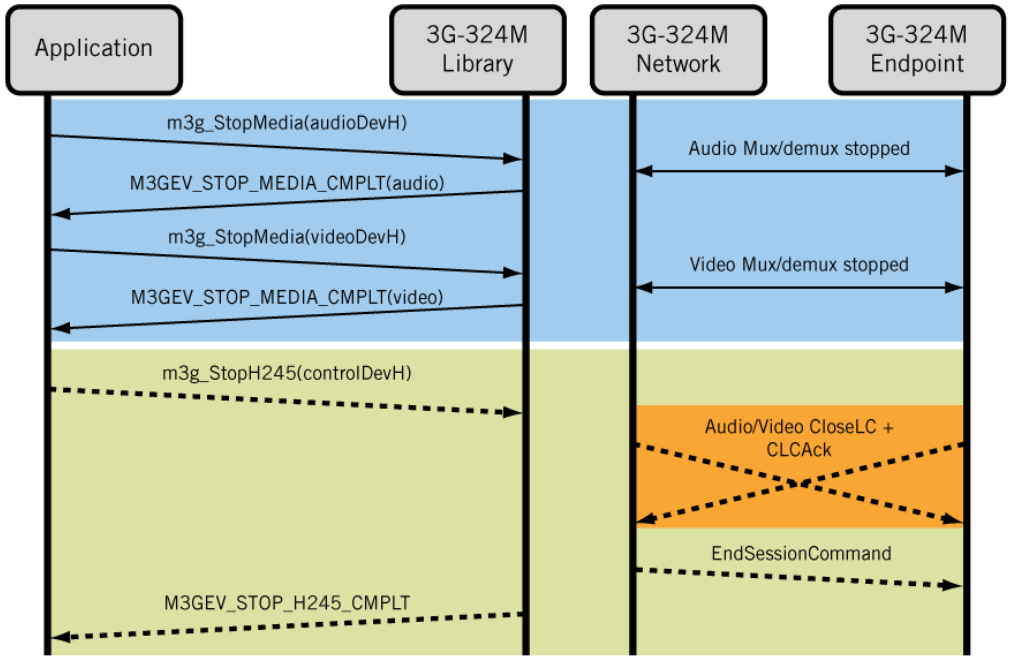
Call **m3g_StopH245( )** to terminate and release all H.245 resources. If an H.245 EndSession message was not received prior to this point from the remote endpoint, a call to **m3g_StopH245( )** triggers the firmware to close logical channels and send an EndSession message to the remote 3G endpoint indicating that the session has ended and will be terminated without further communication.

An M3GEV_ENDSESSION_RCVD event may be returned to the application to signify that the remote endpoint terminated the H.245 session. An EndSession message must be the last 3G-324M message sent from the remote endpoint signifying that the H.245 session has ended and no more communication should be expected. If an EndSession message is returned to the 3G-324M protocol stack, call **m3g_StopH245( )** to terminate the local session.

Note: It is not necessary to call **m3g_CloseLC( )** to initiate closure of a logical channel. This task is performed as part of **m3g_StopH245( )** functionality.

Figure 15 illustrates the terminate session sequence.

Figure 15. Terminate Session Sequence



5.9 Disconnect the Bearer Channel

After the 3G-324M session is terminated, a 3G call is ended by disconnecting the bearer channel through the network. In a PSTN network, the bearer channel containing the data stream is disconnected through ISDN or ISUP signaling methods. In an IP network, the call is typically disconnected using BICC.

If a bearer channel is disconnected, closed or corrupted while a 3G-324M session is in progress, the firmware will determine that the frame synchronization has been lost. An M3GEV_FRAMING_LOST event is generated to the application. Subsequently the firmware will internally tear down any active 3G-324M session and present the M3GEV_ENDSESSION_RCVD event for each previously opened logical channel. No matter the disconnect method, you should follow the process described in Section 5.8, "Terminate a 3G-324M Session", on page 50 to ensure that the Dialogic® 3G-324M software system clean-up occurs and that resources are properly deallocated.

5.10 Disconnect Devices

All connections can optionally be disconnected after the 3G-324M session is stopped or can remain connected for the next 3G call established with a remote endpoint. The following topics describe how to disconnect devices:

- Disconnect Media Port Connections
- Disconnect Network Device

5.10.1 Disconnect Media Port Connections

To disconnect audio and video connections between the m3g device and the mm device, for example, call **dev_PortDisconnect( )**, once for each desired transmit media stream. The **dev_PortDisconnect( )** function removes the unidirectional packet connection between one or more transmit ports of one device to the receive port of another device.

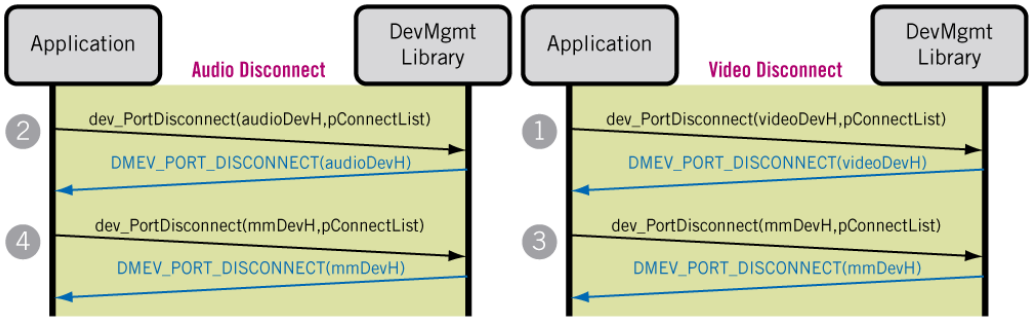
In the following steps, two audio and two video transmit streams are disconnected, one audio Tx stream and one video Tx stream for each device:

1. To disconnect m3g device Tx video stream, call **dev_PortDisconnect( )** with the m3g Tx video port_info list.
2. To disconnect m3g device Tx audio stream, call **dev_PortDisconnect( )** with the m3g Tx audio port_info list.
3. To disconnect mm device Tx video stream, call **dev_PortDisconnect( )** with the mm Tx video port_info list.
4. To disconnect mm device Tx audio stream, call **dev_PortDisconnect( )** with the mm Tx audio port_info list.

Note: Tx and Rx direction represents the internal media stream direction relative to the device, not the direction relative to the 3G network.

Figure 16 illustrates the sequence for disconnecting media port devices.

Figure 16. Disconnect Media Port Devices Sequence



5.10.2 Disconnect Network Device

On the aggregate network side, the m3g device can be disconnected from the PSTN or IP-based network device, representing the ingress or egress of the 3G-324M bearer channel to disconnect the multiplexer output.

In a PSTN network, disconnect the m3g device from the Dialogic® Global Call digital network interface (dti) device using the **dev_Disconnect( )** function. Call this function once with the m3g control handle and once with the dti handle to disconnect the aggregate timeslot connection.

In an IP network, disconnect the m3g device from the ipm device using **dev_PortDisconnect( ).** Call this function once with the m3g control port list and once with the ipm port list, similar to the method described in Section 5.10.1, "Disconnect Media Port Connections", on page 52.

Before closing the devices and terminating the application, devices must be successfully disconnected as indicated by disconnect completion events.

5.11 Close Devices

Before exiting an application, call the appropriate _Close( ) function on all the devices that were opened by the application. This terminates the queuing of all library events for the device, deallocates the resources that were reserved for the device, and returns the device to a known terminated state. Call the _Close( ) function only after devices are disconnected and DMEV_DISCONNECT or DMEV_PORT_DISCONNECT events are received.

For every 3G-324M channel being used by the application, follow these steps to properly close devices:

1. To close the m3g control device, call **m3g_Close( )** using the m3g control device handle.
2. To close the m3g audio device, call **m3g_Close( )** using the m3g audio device handle.
3. To close the m3g video device, call **m3g_Close( )** using the m3g video device handle.

4. To close the m3g board device which may have been opened to configure system-wide settings, call **m3g_Close( )** using the m3g board device handle.
5. Close all other devices in use, such as mm, ipm, and cnf devices using the appropriate functions.

5.12 Exit the application

To exit the application, call **m3g_Stop( )**. This function stops the Dialogic® 3G-324M library and releases all allocated resources. The M3G_SUCCESS event is generated after the 3G-324M library is successfully stopped.

Interoperability and Compliance Information 6

This chapter provides interoperability guidelines for the Dialogic® 3G-324M (m3g) device and compliance information for the 3GPP 3G-324M Technical Specification.

6.1 Interoperability Guidelines

The Dialogic® 3G-324M (m3g) device should be compatible with a handset that supports at least one entry from every item in the following list. Included in the list is a recommended usage that will help to insure interoperability with the maximum number of handsets.

MuxLevel

L0, L1, L2, L2 with optional headers, L3

Recommended usage: L2

SRP (Simple Retransmission Protocol)

NSRP (Numbered SRP) with multiple messages per PDU, WNSRP (Windowed Numbered SRP) with multiple messages per PDU, NSRP without multiple messages per PDU, WNSRP without multiple messages per PDU

Recommended usage: WNSRP with multiple messages per PDU

Media Establishment Procedures

MONA-MPC, MONA-APC, LC with Early MES, LC Legacy

Recommended usage: MONA-MPC

Audio Adaptation Layer

AL1 framed (non-segmented), AL2 with sequence numbers (non-segmented), AL2 without sequence numbers (non-segmented), AL3 as unidirectional/reverse data as null (non-segmented)

Recommended usage: AL2 with sequence numbers

Note: Some handsets that support AL3 do not support reverse null data. It is strongly suggested that the AL3 capability not be advertised in the TCS message. If AL3 is advertised, the TerminalType must be set to a large value to insure that the m3g device is the H.245 master.

Video Adaptation Layer

AL1 framed (segmented), AL2 with sequence numbers (segmented), AL2 without sequence numbers (segmented), AL3 as unidirectional/reverse data as null (segmented)

Recommended usage: AL2 without sequence numbers

Note: See note about AL3 under Audio Adaptation Layer.

Video Codec

H.263 QCIF/SQCIF as profile 0 level 20, MPEG-4 QCIF as simple profile level 0

Recommended usage: H.263 QCIF

Note: The Dialogic® software cannot generate MPEG-4 with data partitioning in the transmit stream when transcoding, and will not interoperate with a handset that requires this video format.

Audio Codec

AMR-NB 1 frame per SDU, G.723.1 2 frames per SDU

Recommended usage: AMR-NB

Maximum ALxSDU size

1024

Recommended usage: 512

Maximum average video bit rate

45 kbps

Recommended usage (transcoding): 40 kbps

Maximum average audio bandwidth

12.2 kbps

Recommended usage: 12.2 kbps

Further recommended settings to improve interoperability

Terminal Type: 192
Video AL2 SDU Size: 256
Frame Rate (transcoding): 6 fps

6.2 Statements of Compliance

The following is a Statement of Compliance for Dialogic® 3G-324M software with the 3GPP TS 26.111 V7.1.0, Codec for Circuit-Switched Multimedia Telephony Service, Modifications to H.324. Release 7.

Table 1. Statement of Compliance with 3GPP TS 26.111 V7.1.0

Section	Compliance
6.1	Compliant
6.4	Partially compliant: Optional H.223 adaptation layers AL1M, AL2M, and AL3M not supported
6.5	Partially compliant: H.245 bidirectional LC procedures not supported
6.6	Partially compliant: Optional MPEG-4 codec does not include RM, DP, HEC, RVLC codec options when transcoding Optional H.264 codec not supported
6.6.1	Compliant

Table 1. Statement of Compliance with 3GPP TS 26.111 V7.1.0 (Continued)

Section	Compliance
6.6.2	N/A (H.264 codec)
6.7	Partially compliant: Optional AMR-WB not supported
6.8	N/A (data channels)
7	Compliant
8	Compliant
9	N/A
10	N/A

The following is a Statement of Compliance for Dialogic® 3G-324M software with the 3GPP TR 26.911, Codec for Circuit-Switched Multimedia Telephony Service, Terminal Implementer's Guide Release 4.

Table 2. Statement of Compliance with 3GPP TR 26.911

Section	Compliance
5	Compliant
5.1	Compliant
6	Partially compliant: Bidirectional LC establishment not supported Optional LC replacement procedures not supported
6.1	N/A
7	Compliant
7.1	Compliant
7.2	Partially compliant: H.263 Appendix I, J, K, T not supported
7.3	N/A (data channels)
7	Compliant
8	Compliant
9	N/A
10	N/A
11	Compliant
12	N/A
13	N/A
14	N/A
15	N/A

Video Quality Considerations 7

This chapter describes points to consider when working with video files, especially as they relate to video quality.

For 3G-324M connections, the maximum total available bandwidth for audio, video, and H.245 control messages is limited to 64 kbps. Therefore, in order for the 3G endpoint to receive reasonable audio and video quality, it is recommended that the peak bit rate for the video stream transmitted should not exceed 30 kbps when the audio codec is AMR-NB, and should not exceed 35 kbps when the audio codec is G.723.1. It is also critical that the video stream is generated using the constant bit rate (CBR) mode as opposed to the variable bit rate (VBR) mode and that video frame sizes not exceed recommended maximum values. This is necessary to insure smooth flowing, non-jittery video when displayed on a 3G handset. Since the transport total bandwidth is limited to a maximum 64 kbps (8 K bytes per second), a video frame size of 4 K bytes, for example, would take a minimum of a half second to be transmitted. At a frame rate of 10 fps, a display frame update is expected approximately every 100 ms. Therefore, the display time of large frames would be delayed and would be seen as stalled and/or jittery, non-smooth flowing video.

When video transcoding is used, along with the proper settings, the encoder will insure that the video bit stream is appropriate for the transport. When video is played natively using native connections, the source file that is being played must have the appropriate video stream characteristics to match the needs of the transport and the 3G handset. (Transcoding connection type is set in the Dialogic® Device Management API library, DM_PORT_CONNECT_INFO structure.)

Several parameters can affect the bit rate of a given video stream. Depending on the content creation tool, you can typically control the following parameters that trade off bit rate versus quality:

- Image Size: it is recommended that the image size should be limited to QCIF or smaller.
- Frame Rate: it is recommended that the frame rate should not exceed 10 frames/sec.
- Bit Rate: as documented above.
- Maximum Packet (or Frame) Size: if this option is available, it is recommended that the maximum video frame or packet size should not exceed 1000 bytes.
- Maximum time between I-Frames: it is recommended that the maximum time between I-frames be set to no greater than 5 seconds.

Data Structure Considerations 8

This chapter describes points to consider when working with data structures. Topics include:

8.1 Using Inline Functions

Some data structures in the 3G-324M library have an associated inline function. Use the inline function, where available, to initialize the fields of the data structure, including the version number field. For example, use the INIT_M3G_H223_SESSION( ) inline function to initialize the fields of the M3G_H223_SESSION structure.

8.2 Handling the Version Number

Some data structures in the 3G-324M library have a version number field. This version number is used to ensure that an application is binary compatible with future changes to this data structure. This field should be set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

Building Applications 9

This chapter provides information for those choosing to build applications using the Dialogic® 3G-324M API library. The following topics are discussed:

9.1 Compiling and Linking

An application that uses the Dialogic® 3G-324M API must include references to the Dialogic® 3G-324M API header files and must include the appropriate library files. The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

9.1.1 Include Files

The following header files must be included in the application code in the order shown prior to calling the library functions:

srllib.h
Contains function prototypes and equates for the Dialogic® Standard Runtime Library.

Note: *srllib.h* must be included in code before all other Dialogic® header files.

m3glib.h
The primary header file for the Dialogic® 3G-324M library. Contains function prototypes and symbolic defines.

m3gerrs.h
Contains enumeration constants for Dialogic® 3G-324M error codes.

m3gevts.h
Contains symbolic constants for Dialogic® 3G-324M event codes.

9.1.2 Required Libraries

The following library files must be linked in the order shown:

libsrl.so
Dialogic® Standard Runtime Library API file. Required in all applications. Specify `-lsrl` in makefile.

libm3g.so
Dialogic® 3G-324M library file. Required in a 3G-324M application. Specify `-lm3g` in makefile.

Note: When compiling an application, you must list Dialogic® libraries before all other libraries such as operating system libraries.

By default, the library files are located in the directory given by the INTEL_DIALOGIC_LIB environment variable.

9.2 Variables for Compiling and Linking

The following variables provide a standardized way of referencing the directories that contain header files and shared objects:

INTEL_DIALOGIC_INC
Variable that points to the directory where header files are stored.

INTEL_DIALOGIC_LIB
Variable that points to the directory where shared library files are stored.

These variables are automatically set at login and should be used in compiling and linking commands.

Debugging 10

This chapter provides information about debugging Dialogic® 3G-324M software applications:

10.1 Trace Utilities

Runtime tracing on a per channel or board basis is supported. This feature allows you to set tracing of the following:

- H.245 messaging (with textual decode)
- raw binary H.223 multiplexed bitstreams
- raw binary audio streams
- raw binary video streams
- call statistics

Logging is directed to a user-specified file. The default location for log files is /usr/dialogic/data.

To initiate and configure tracing, use **m3g_StartTrace( )**. The logging levels used in **m3g_StartTrace( )** are configured by setting the respective bits in the bitmask field in the M3G_TRACE_INFO structure; see this data structure description for details. To stop tracing, use **m3g_StopTrace( )**.

10.1.1 Parser Utility

A post-processing 3G log file parser is provided to parse log files created via **m3g_StartTrace( )**. The parser utility is called m3g_parser and is used as follows from the command line:

```
m3g_parser <logfileName>
```

The tool takes one mandatory argument which is the filename of the resulting log file. After executing the parser, the tool parses various 3G-specific trace entries into separate files for each device. A log file is created for each device which has logged subject-related entries using the following naming convention (where n is the m3g device number):

h245_n.txt
transmitted and received, timestamped H.245 messages in ASCII format for device n

h223tx_n.bin
transmitted H.223 multiplex bit stream in binary format for device n

h223rx_n.bin
received H.223 multiplex bit stream in binary format for device n

audiotx_n.bin
transmitted audio bit stream in binary format which is subsequently multiplexed for device n

audiorx_n.bin
received audio bit stream in binary format after demultiplexing via device n

videotx_n.bin
transmitted video bit stream in binary format which is subsequently multiplexed for device n

audiorx_n.bin
received video bit stream in binary format after demultiplexing via device n

stats_n.txt
call signaling and media statistics recorded for device n. Statistics include counts of multiplex and media packets sent and received, stuffing bytes, and errors.

10.2 Call Statistics

Call statistics provide statistics such as call duration, a count of the transmitted and received packets, total bytes, CRC and packet errors for the H.223 multiplex bit stream, audio and video and for the session.

To use call statistics per call, enable the event bit M3G_CALL_STATISTICS_EVT_TYP in **m3g_EnableEvents( )**. The default is disabled.

When enabled, on the completion of every 3G-324M session, an M3GEV_CALL_STATISTICS event is queued to the application. The event includes call statistics; see M3G_CALL_STATISTICS data structure for details.

The M3GEV_CALL_STATISTICS event is queued after the 3G-324M session is terminated immediately following the queuing of the M3GEV_STOP_H245_CMPLT event for **m3g_StopH245( )**.

Function Summary by Category 11

This chapter describes the categories into which the Dialogic® 3G-324M API functions can be logically grouped. Topics include:

11.1 System Control Functions

The following functions provide device and library management capabilities:

m3g_Close( )
closes the specified device

m3g_DisableEvents( )
disables one or more unsolicited events

m3g_EnableEvents( )
enables one or more unsolicited events

m3g_GetParm( )
gets the current parameter settings for the specified device

m3g_GetUserInfo( )
gets a user-defined device handle for an SRL device

m3g_Open( )
opens the specified device

m3g_OpenEx( )
opens the specified device in synchronous or asynchronous mode

m3g_Reset( )
resets open devices that were improperly closed

m3g_SetParm( )
sets a parameter for the specified device

m3g_Start( )
starts and initializes the 3G-324M library

m3g_Stop( )
stops the 3G-324M library and releases all allocated resources

11.2 H.245 Control Functions

The following functions manage H.245 multimedia exchange messages and communication:

m3g_CloseLC( )
initiates closure of specified logical channel number

m3g_GetLocalCaps( )
gets the default capabilities supported by the specified device

m3g_GetMatchedCaps( )
gets common capabilities between the local and the remote endpoints

m3g_OpenLC( )
sends an OpenLogicalChannel request

m3g_RespondToOLC( )
responds to an OpenLogicalChannel request

m3g_SendH245MiscCmd( )
sends the specified H.245 MiscellaneousCommand message to the remote endpoint

m3g_SendH245UII( )
sends DTMF digits in an H.245 UserInputIndication message to the remote endpoint

m3g_SetTCS( )
sets the local set of terminal capabilities in the H.245 TerminalCapabilitySet table

m3g_SetVendorId( )
configures information elements to be encoded in the H.245 VendorIdentification indication message

m3g_StartH245( )
initiates the H.223 multiplex and demultiplex

m3g_StopH245( )
terminates the H.245 session

11.3 Data Flow Functions

The following functions manage the data flow between the media device and the H.223 multiplex/demultiplex:

m3g_ModifyMedia( )
modifies the streaming characteristics to and from the specified media device

m3g_StartMedia( )
starts the media stream between the media device and the H.223 multiplex/demultiplex

m3g_StopMedia( )
stops the media stream between the media device and the H.223 multiplex/demultiplex

11.4 Utility Functions

The following utility functions are available to configure and enable additional features:

m3g_StartTrace( )
initiates and configures 3G-324M tracing to a user-specified log file

m3g_StopTrace( )
stops 3G-324M tracing previously specified for a device or devices

Function Information 12

This chapter provides an alphabetical reference to the functions in the Dialogic® 3G-324M API library. A general description of the function syntax convention is provided before the detailed function information.

All function prototypes are in the *m3glib.h* header file.

12.1 Function Syntax Conventions

The Dialogic® 3G-324M API functions typically use the following format:

```
int m3g_Function (deviceHandle, parameter1, parameter2, ... parametern)
```

where:

int
: represents an integer return value that indicates if the function succeeded or failed. Possible values are:
  - 0 if the function succeeds
  - <0 if the function fails

m3g_Function
: represents the name of the function

deviceHandle
: refers to an input field representing the type of device handle (board, control, audio, video)

parameter1, parameter2, ... parameter*n*
: represent input or output fields

m3g_Close( )

Name: int m3g_Close (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board, control, audio or video device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: synchronous

Description

The **m3g_Close( )** function closes the handle for a device that was previously opened and terminates the queuing of all 3G-324M library events associated with this handle.

Parameter	Description
deviceHandle	specifies an SRL handle to a board, control, audio or video device obtained from a previous open

This function is only supported in synchronous mode. The function returns M3G_SUCCESS if the specified device was successfully closed; otherwise, it returns M3G_ERROR.

Cautions

Audio, video and control devices associated with an H.223 bearer channel should be closed only after terminating the associated H.245 session using **m3g_StopH245( )**.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
```

```
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) Assume global device table defined elsewhere            */
int CloseDevs(int numDevices)
{
   int devNum;

   for(devNum=1; devNum<=numDevices; devNum++)
   {
      /* Close control device for 3G-324M endpoint */
      if (M3G_ERROR == m3g_Close(devTbl[devNum].cDev))
      {
         log("Error:  m3g_Close(%s)failed - %s\n",
              ATDVNAMEP(devTbl[devNum].cDev), ATDV_ERRMSGP(devTbl[devNum].cDev));
         /* handle error… */
      }

      /* Close audio device for 3G-324M endpoint */
      if (M3G_ERROR == m3g_Close(devTbl[devNum].aDev))
      {
         log("Error:  m3g_Close(%s)failed - %s\n",
              ATDVNAMEP(devTbl[devNum].aDev), ATDV_ERRMSGP(devTbl[devNum].aDev));
         /* handle error… */
      }

      /* Close video device for 3G-324M endpoint */
      if (M3G_ERROR == m3g_Close(devTbl[devNum].vDev))
      {
         log("Error:  m3g_Close(%s)failed - %s\n",
              ATDVNAMEP(devTbl[devNum].vDev), ATDV_ERRMSGP(devTbl[devNum].vDev));
         /* handle error… */
      }
   }
   /* Close board device */
   if (M3G_ERROR == m3g_Close(boardDev))
   {
      log("Error:  m3g_Close(m3gB1)failed -%s\n",
           ATDV_ERRMSGP(boardDev));
      /* handle error… */
   }

   return SUCCESS;
} /* End of CloseDevs */
```

See Also

- **m3g_Open( )**
- **m3g_StopH245( )**

m3g_CloseLC( )

Name:	int m3g_CloseLC (deviceHandle, lcn, reason)	
Inputs:	SRL_DEVICE_HANDLE deviceHandle	• control device handle
	M3G_LOGICAL_CHANNEL_NUMBER lcn	• number of logical channel to be closed
	M3G_E_REQ_CHAN_CLOSE_REASON reason	• reason
Returns:	M3G_SUCCESS if successful M3G_ERROR on failure	
Includes:	srllib.h m3glib.h m3gevts.h m3gerrs.h	
Category:	H.245 Control	
Mode:	asynchronous	

Description

The **m3g_CloseLC( )** function initiates closure of the specified logical channel number by sending a message to the remote 3G-324M endpoint. If a forward logical channel is specified in **lcn**, a CloseLogicalChannel message is sent. If a reverse logical channel is specified in **lcn**, a RequestChannelClose message is sent.

The function completes only after receiving a CloseLogicalChannelAck or RequestChannelCloseAck response from the remote 3G-324M endpoint as indicated via the M3GEV_CLOSE_LC_CMPLT event. Upon receiving this event, use the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to retrieve the void data buffer embedded within the event and cast it as a pointer to an unsigned short to decode the logical channel number that was successfully closed.

When an incoming CloseLogicalChannel command or RequestChannelClose request is received from the remote 3G-324M endpoint, an M3GEV_REMOTE_CLOSE_LC_RCVD event is queued to the application which includes the logical channel number. Upon receiving this event, use the Standard Runtime Library function **sr_getevtdatap( )** to retrieve the void data buffer embedded within the event and cast it as a pointer to an M3G_REMOTE_CLOSED_LC structure to obtain the closed logical channel number and the reason.

Note that incoming CloseLogicalChannel commands are always implicitly and automatically responded to via a CloseLogicalChannelAck response by the 3G-324M protocol stack. Similarly, RequestChannelClose requests are always implicitly and automatically responded to via a RequestChannelCloseAck response. The application is only responsible for properly re-routing the associated media stream from the H.223 multiplex aggregate.

Note that as the **m3g_StopMedia( )** function terminates the H.223 multiplexing and de-multiplexing, it is possible for the application to maintain the packet stream connections between

the H.223 multiplex and the actual audio and video media devices for the next 3G-324M multiplexed call.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
lcn	specifies the logical channel number to be closed
reason	M3G_E_REQ_CHAN_CLOSE_REASON enumeration type specifying H.245 reason to be included in the CloseLogicalChannel or RequestChannelClose message. For more information about possible values, see the M3G_REMOTE_CLOSED_LC structure description.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_CLOSE_LC_CMPLT
: Indicates a CloseLogicalChannelAck or RequestChannelCloseAck message has been successfully received from the remote 3G-324M endpoint acknowledging the CloseLogicalChannel command or RequestChannelClose request.

M3GEV_CLOSE_LC_FAIL
: Indicates either a local failure to send the CloseLogicalChannel request or no CloseLogicalChannel response was received. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

It is invalid to call this function with a board, audio or video device type handle.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.


/* Preconditions: 1) 3G-324M Library H.245 Forward Logical Channel has already been      */
/*                   established (not shown).                                           */
/*                2) Call associated with bearer channel is disconnected (not shown).    */

int handleDisconnectedCall(MYDEV * pMyDev)
```

```
{
   /* Disconnect and stop the media (not shown): */
   disconnectAndStopMedia(pMyDev);

   /* Close audio forward logical channel */
   if (M3G_ERROR == m3g_CloseLC(pMyDev->controlDevH, pMyDev->audioLCN,
            M3G_E_REQ_CHAN_CLOSE_NORMAL))
   {
      log("Error:  m3g_CloseLC(%s)failed – %s\n",
           ATDVNAMEP(pMyDev->controlDevH), ATDV_LASTERR(pMyDev->controlDevH));
      /* handle error… */
   }

   /* Close video forward logical channel */
   if (M3G_ERROR == m3g_CloseLC(pMyDev->controlDevH, pMyDev->videoLCN,
            M3G_E_REQ_CHAN_CLOSE_NORMAL))
   {
      log("Error:  m3g_CloseLC(%s)failed – %s\n",
           ATDVNAMEP(pMyDev->controlDevH), ATDV_LASTERR(pMyDev->controlDevH));
      /* handle error… */
   }

   /* Stop the H.245 Session (not shown): */
   stopH245(pMydev);

}  /* End of handleDisconnectedCall */
.
.
.
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
           process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_CloseLC termination: */
        case M3GEV_CLOSE_LC_CMPLT:
        {
           /* Assume application defined its device structure: */
           MYDEV * pMyDev;
           M3G_LOGICAL_CHANNEL_NUMBER lcn = *(M3G_LOGICAL_CHANNEL_NUMBER *) pSRLEvtData;
           m3g_GetUserInfo(devH, &pMyDev);

           /* Determine whether lcn is for audio or video: */
           if(AUDIO == getLCNMediaType(pMyDev, lcn))
           {
               pMyDev->isAudioFwdOLCAcked = false;
               pMyDev->fwdAudioLCN = 0;
```

```
            }
            else   /* else video: */
            {
                pMyDev->isVideoFwdOLCAcked = false;
                pMyDev->fwdVideoLCN = 0;
            }

            /* If both audio and video CLCAcks received: */
            if ((!pMyDev->isAudioFwdOLCAcked) && (!pMyDev->isVideoFwdOLCAcked))
            {
               /* Stop the H.245 Session (not shown): */
               stopH245(pMydev);
            }
      break;
   }

        /* m3g_CLoseLC Failure indication: */
        case M3GEV_CLOSE_LC_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_CLOSE_LC_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
      }
   }
```

See Also

- **m3g_OpenLC( )**

m3g_DisableEvents( )

Name: int m3g_DisableEvents (deviceHandle, eventBitmask)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board or control device handle

unsigned int eventBitmask
- bitmask of unsolicited events to disable

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_DisableEvents( )** function disables one or more unsolicited events for a board device or for a control device. For more information on device types, see the Device Types section in the **m3g_Open( )** function reference.

Not all unsolicited events can be masked and disabled. The following unsolicited events can be masked and disabled:

- M3GEV_H245_UII_RCVD (enabled by default)
- M3GEV_H245_MISC_CMD_RCVD (enabled by default)
- M3GEV_H245_MSD_EVT (disabled by default)
- M3GEV_H245_MES_EVT (disabled by default)
- M3GEV_REMOTE_VENDORID_RCVD (disabled by default)
- M3GEV_MONA_PREF_MSG_RCVD (disabled by default)
- M3GEV_SEND_MONA_PREF_MSG (disabled by default)
- M3GEV_CALL_STATISTICS (disabled by default)
- M3GEV_IFRAME_RCVD (disabled by default)

To change default settings, use **m3g_EnableEvents( )** and **m3g_DisableEvents( )**. See Chapter 13, "Events" for more information on these events.

If an event is enabled/disabled for a board device, the specified event is enabled/disabled for all control channels opened on that board. If an event is enabled/disabled for a control device, the specified event is only enabled/disabled for that individual control device.

This function is only supported in asynchronous mode.

Parameter	Description
deviceHandle	specifies an SRL handle to a board device or a control device

Parameter	Description
eventBitMask	a bitmask of bit field constants specifying unsolicited notification event types to disable. The bitmask can be any combination of the bit field constants. See **m3g_EnableEvents( )** for a list of values.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.                    */
/*                2) m3g_Open( ) has completed opening the board or a control device handle. */


int disableEvents(int devH)
{
   /* Disable all maskable 3G-324M events on specified board or control device:  */

   /* Only M3GEV_H245_MSD_EVT and M3GEV_H245_MES_EVT events are disabled by default */
   unsigned int evBitMask = (M3G_H245_UII_EVT_TYP |
                             M3G_H245_FASTUPDATE_EVT_TYP |
                             M3G_H245_TEMP_SPAT_TRDFF_EVT_TYP |
                             M3G_H245_VIDEO_FREEZE_EVT_TYP |
                             M3G_H245_SYNC_GOB_EVT_TYP |
                             M3G_LC_INACTIVE_EVT_TYP |
                             M3G_SKEW_INDICATION_EVT_TYP);
   if (M3G_ERROR == m3g_DisableEvents(devH, evBitMask))
   {
      log("Error:  m3g_DisableEvents(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of disableEvents */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
```

```
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_DisableEvents termination: */
        case M3GEV_DISABLE_EVENTS_CMPLT:
            log("M3GEV_DISABLE_EVENTS_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            /* Device is ready for 3G-324M processing */
            deviceIsReady(devH);  /* proceed with 3G-324M processing (not shown)*/
            break;

        /* m3g_DisableEvents Failure indication: */
        case M3GEV_DISABLE_EVENTS_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_DISABLE_EVENTS_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_EnableEvents( )**

m3g_EnableEvents( )

Name: int m3g_EnableEvents (deviceHandle, eventBitmask)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board or control device handle

unsigned int eventBitmask
- bitmask of unsolicited events to enable

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_EnableEvents( )** function enables one or more unsolicited events for a board device or for a control device. For more information on device types, see the Device Types section in the **m3g_Open( )** function reference.

Not all unsolicited events can be masked and enabled. The following unsolicited events can be masked and enabled:

- M3GEV_H245_UII_RCVD (enabled by default)
- M3GEV_H245_MISC_CMD_RCVD (enabled by default)
- M3GEV_H245_MSD_EVT (disabled by default)
- M3GEV_H245_MES_EVT (disabled by default)
- M3GEV_REMOTE_VENDORID_RCVD (disabled by default)
- M3GEV_MONA_PREF_MSG_RCVD (disabled by default)
- M3GEV_SEND_MONA_PREF_MSG (disabled by default)
- M3GEV_CALL_STATISTICS (disabled by default)
- M3GEV_IFRAME_RCVD (disabled by default)

To change default settings, use **m3g_EnableEvents( )** and **m3g_DisableEvents( )**. See Chapter 13, "Events" for more information on these events.

If an event is enabled/disabled for a board device, the specified event is enabled/disabled for all control channels opened on that board. If an event is enabled/disabled for a control device, the specified event is only enabled/disabled for that individual control device.

This function is only supported in asynchronous mode.

Parameter	Description
deviceHandle	specifies an SRL handle to a board device or a control device
eventBitMask	a bitmask of bit field constants specifying unsolicited notification event types to enable. The bitmask can be any combination of the following bit field constants: • M3G_H245_UII_EVT_TYP – M3GEV_H245_UII_RCVD event notification of incoming H.245 UII • M3G_H245_FASTUPDATE_EVT_TYP – M3GEV_H245_MISC_CMD_RCVD event notification of incoming H.245 MiscellaneousCommand indication messages of type videoFastUpdate, videoFastUpdateGOB, or videoFastUpdateMB • M3G_H245_TEMP_SPAT_TRDFF_EVT_TYP – M3GEV_H245_MISC_CMD_RCVD event notification of incoming H.245 MiscellaneousCommand indication messages of type videoTemporalSpatialTradeoff • M3G_H245_VIDEO_FREEZE_EVT_TYP – M3GEV_H245_MISC_CMD_RCVD event notification of incoming H.245 MiscellaneousCommand indication messages of type videoFreeze • M3G_MES_EVTS_EVT_TYP – verbose notification of H.245 Multiplex Entry Send message transactions via the M3GEV_H245_MES_EVT event • M3G_VERBOSE_MSD_EVT_TYP – verbose notification of H.245 MasterSlaveDetermination message transactions via the M3GEV_H245_MSD_EVT event ***Note:*** This setting does not impact the notification of M3GEV_MSD_ESTABLISHED or M3GEV_MSD_FAILED events which are always enabled and active. • M3G_REMOTE_VENDORID_EVT_TYP – M3GEV_REMOTE_VENDORID_RCVD event notification of remote vendor and product information in incoming H.245 VendorIdentification message • M3G_MONA_PREF_MSG_EVT_TYP – M3GEV_MONA_PREF_MSG_RCVD event notification of the first incoming MONA preference message and M3GEV_SEND_MONA_PREF_MSG event notification of the last MONA preference message sent • M3G_CALL_STATISTICS_EVT_TYP – M3GEV_CALL_STATISTICS event notification of statistics upon call termination. Statistics include media and H.223 data transmission, transmission errors, and call duration.

Parameter	Description
eventBitMask (continued)	• M3G_IFRAME_EVT_TYP – Controls M3GEV_IFRAME_RCVD event notification of incoming Iframes detected inband from either the remote 3G-324M endpoint or from the device connected to the m3g device (IPM, MM, etc.). This event only applies to H.264 codec. ***Note:*** To receive notification, the application must enable the event with **m3g_EnableEvents( )** and configure Iframe reporting by setting M3G_E_PRM_IFRAME_NOTIFY_CONTROL_MASK with **m3g_SetParm( )**.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.     */
/*                2) m3g_Open( ) has opened the devH board or a control device handle. */

int enableEvents(int devH)
{
   /* Enable all maskable 3G-324M events on specified board or control device. */

   /* Only M3GEV_H245_MSD_EVT and M3GEV_H245_MES_EVT events are disabled on default */
   unsigned int evBitMask = (M3G_VERBOSE_MSD_EVT_TYP | M3G_MES_EVTS_EVT_TYP);
   if (M3G_ERROR == m3g_EnableEvents(devH, evBitMask))
   {
      log("Error:  m3g_EnableEvents(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of enableEvents */
.
.
.

/* SRL event handler: */
   for (;;)
   {
```

```
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_EnableEvents termination: */
        case M3GEV_ENABLE_EVENTS_CMPLT:
            log("M3GEV_ENABLE_EVENTS_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            /* Device is ready for 3G-324M processing */
            deviceIsReady(devH);  /* proceed with 3G-324M processing (not shown)*/
            break;

        /* m3g_EnableEvents Failure indication: */
        case M3GEV_ENABLE_EVENTS_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_ENABLE_EVENTS_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_DisableEvents( )**

m3g_GetLocalCaps( )

Name: int m3g_GetLocalCaps (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control, audio or video device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_GetLocalCaps( )** function retrieves the default capabilities supported by the device type specified in the **deviceHandle** parameter. The type of device and its capabilities are:

- control - H.223 multiplex capabilities
- audio - G.723.1 and AMR capabilities
- video - H.263, H.264, and MPEG-4 capabilities

Call this function for each device type in use.

Parameter	Description
deviceHandle	specifies an SRL handle to a control, audio or video device. The resulting local capabilities returned in the M3G_CAPS_LIST structure are subject to the type of device. A control, audio or video device returns H.223 capability, audio capability, or video capability information, respectively.

This function is only supported in asynchronous mode.

Upon successful completion via receipt of the termination event, the M3G_CAPABILITY union is returned via the M3G_CAPS_LIST structure. The M3G_CAPABILITY union elements are of the applicable data type, M3G_ H223_CAPABILITY, M3G_AUDIO_CAPABILITY, or M3G_VIDEO_CAPABILITY, respectively. You can use the default settings in the M3G_CAPABILITY union, or modify the settings when calling **m3g_SetTCS( )** to specify which H.223 multiplex, audio and video capabilities will be subsequently used in any H.245 TerminalCapabilitySet message sent to the remote 3G-324M endpoint.

After the function returns M3G_SUCCESS, the application must wait for the M3GEV_GET_LOCAL_CAPS_CMPLT event. Once the event has been returned, use the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to retrieve the void data buffer

embedded within the event and cast it as M3G_CAPS_LIST to decode the local H.223 multiplex, audio or video capabilities within each M3G_CAPABILITY.

Termination Events

M3GEV_GET_LOCAL_CAPS_CMPLT
: Indicates the local capabilities have been successfully retrieved. The resulting M3G_CAPS_LIST structure can be retrieved by calling **sr_getevtdatap( )** to retrieve the data buffer embedded within this event and casting it from data type void pointer to data type M3G_CAPS_LIST structure pointer. The data within this buffer must be processed or copied before the next SRL event is de-queued, at which point this buffer will de-allocated by the Standard Runtime Library.

M3GEV_GET_LOCAL_CAPS_FAIL
: Indicates the local capabilities for the specified device type could not be retrieved successfully. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has been completed for the applicable       */
/*                   control, audio and video device type                    */
/*                3) assumes globally defined devTbl[] exists for all        */
/*                   devices                                                 */
int getDefaultCaps(int devIndex)
{

   M3G_CAPS_LIST * pLocalCaps = &h223Caps;
   /* Retrieve the default H.233 capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].controlDevH))
   {
      log("Error:  m3g_GetLocalCaps(%s) for H.223 failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   /* Retrieve the default audio capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].audioDevH))
```

```
   {
      log("Error:  m3g_GetLocalCaps(%s) for audio failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   /* Retrieve the default video capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].videoDevH))
   {
      log("Error:  m3g_GetLocalCaps(%s) for video failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of getDefaultCaps */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_GetLocalCaps termination: */
        case M3GEV_GET_LOCAL_CAPS_CMPLT:
        {
            /* Assume application defined its device structure: */
            MYDEV * pMyDev;
            M3G_CAPS_LIST * pLocalCaps = pSRLEvtData;

            log("M3GEV_GET_LOCAL_CAPS_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));

            /* Cache appropriate device type (h223, audio, or video) caps in  */
            /* Simultaneous Caps structure to send in TCS: */
            m3g_GetUserInfo(devH, &pMyDev);

            switch (pMyDev->myType)
            {
                case H223TYPE:
                   formatH223CapsInTCS(pMyDev->bearerChannel, pLocalCaps);
                   /* Assume haveLocalCaps is bitmask to identify which caps have been received:
*/
                   pMyDev->bearerChannel.rcvdLocalCaps |= HAVE_H223_CAPS;
                   break;
```

```
                case AUDIOTYPE:
                   formatAudioCapsInTCS(pMyDev->bearerChannel, pLocalCaps);
                   /* Assume haveLocalCaps is bitmask to identify which caps have been received:
*/
                   pMyDev->bearerChannel.rcvdLocalCaps |= HAVE_AUDIO_CAPS;
                   break;
                case VIDEOTYPE:
                   formatVideoCapsInTCS(pMyDev->bearerChannel, pLocalCaps);
                   /* Assume haveLocalCaps is bitmask to identify which caps have been received:
*/
                   pMyDev->bearerChannel.rcvdLocalCaps |= HAVE_VIDEO_CAPS;
                   break;
            }
            /* If received all local capabilities associated with bearer channel: */
            if ((HAVE_VIDEO_CAPS | HAVE_VIDEO_CAPS | HAVE_VIDEO_CAPS) ==
                (pMyDev->bearerChannel.rcvdLocalCaps))
            {
                /* Set Default TCS: */
                setDefaultTCS(pMyDev->bearerChannel);
            }
            break;
        }

        /* m3g_GetLocalCaps Failure indication: */
        case M3GEV_GET_LOCAL_CAPS_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_GET_LOCAL_CAPS_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

       /*
        .
        . Other events not shown…
        .
        */

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_SetTCS( )**

m3g_GetMatchedCaps( )

Name: int m3g_GetMatchedCaps (deviceHandle, pMatchedCapsList)

Inputs:

SRL_DEVICE_HANDLE deviceHandle	• control, audio or video device handle
M3G_CAPS_LIST *pMatchedCapsList	• pointer to M3G_CAPS_LIST structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: synchronous

Description

The **m3g_GetMatchedCaps**( ) function retrieves common capabilities between the remote and the local 3G-324M endpoints for the device type specified in the **deviceHandle** parameter. The type of device and its capabilities are:

- control - H.223 multiplex capabilities
- audio - G.723.1 and AMR capabilities
- video - H.263 and MPEG-4 capabilities

Call this function for each device type in use.

Parameter	Description
deviceHandle	specifies an SRL handle to a control, audio, or video device
pMatchedCapsList	points to the M3G_CAPS_LIST structure. The resulting matched capabilities returned in the M3G_CAPS_LIST structure are subject to the type of device. A control, audio or video device returns H.223 capability, audio capability, or video capability information, respectively. The M3G_CAPABILITY union elements are of the applicable data type, M3G_ H223_CAPABILITY, M3G_AUDIO_CAPABILITY, or M3G_VIDEO_CAPABILITY, respectively.

Only call this function after H.245 MasterSlaveDetermination transactions and H.245 TerminalCapabilitySet transactions have completed in each direction with the remote 3G-324M endpoint.

After successful completion of MasterSlaveDetermination transactions, the application should receive the M3GEV_MSD_ESTABLISHED event. After remote terminal capabilities are received in a TerminalCapabilitySet message from the remote 3G-324M endpoint, an M3GEV_REMOTE_TCS_RCVD event is queued to the application. When local terminal capabilities have been positively acknowledged via the remote, an M3GEV_LOCAL_TCS_ACKD event is queued to the application. Therefore, only call **m3g_GetMatchedCaps( )** after the application receives these three events indicating that the capabilities have been successfully exchanged between the local and remote 3G-324M endpoints.

The **m3g_GetMatchedCaps( )** function populates the M3G_CAPS_LIST structure with capabilities that are supported by both the remote and local endpoints.

Upon successful function completion, the application can choose to extract any of the transmit M3G_CAPABILITY unions returned in the M3G_CAPS_LIST structure. The audio and video capabilities returned in the M3G_CAPS_LIST array are listed in decreasing order of preference by the endpoint deemed master in H.245 MasterSlaveDetermination. Thus, for H.245 compliant behavior, the first audio or video transmit capability should be used in opening audio or video logical channels.

The application can then optionally use these M3G_CAPABILITY unions as is, or after modification, in a subsequent call to **m3g_OpenLC( )**. The **m3g_OpenLC( )** function opens an H.245 logical channel for the specified audio or video stream via an OpenLogicalChannel message sent to the remote 3G-324M endpoint. The audio and video capabilities may be modified in the call to **m3g_OpenLC( )**. However, modifying existing fields within the audio and video capability array elements may lead to undefined behavior.

This function is only supported in synchronous mode. The function returns M3G_SUCCESS if the specified common capability types were successfully retrieved; otherwise, it returns M3G_ERROR.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
```

```
/*                2) m3g_Open( ) has completed opening the                  */
/*                   applicable control, audio and or video devices         */
/*                   associated with the 3G-324M bearer channel.            */
/*                3) The control, audio, and or video devices have all been */
/*                   interconnected to their respective network and ipm or  */
/*                   mm devices using the dev_PortConnect( ) or             */
/*                   dev_Connect( )functions.                               */
/*                4) The default simultaneous caps table has been set using */
/*                   the m3g_GetLocalCap( ) and m3g_SetTCS( ) function      */
/*                   (not shown).                                           */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_StartH245 termination: */
        case M3GEV_START_H245_CMPLT:
            log("M3GEV_START_H245_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            /* Device must receive M3GEV_FRAMING_ESTABLISHED before it  */
            /* can participate in MasterSlaveDetermination exchange.    */
            break;

        /* m3g_StartH245 Failure indication: */
        case M3GEV_START_H245_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_START_H245_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        /* Received TCS from remote 3G-324M endpoint:  */
        case M3GEV_REMOTE_TCS_RCVD:
            {
               /* Assume application defined its device structure: */
               MYDEV * pMyDev;

               /* If both local and remote TCS transactions have completed, can */
               /* initiate the opening of logical channels.                     */
```

```
                /* Cache this TCS transaction completion */
                m3g_GetUserInfo(devH, &pMyDev);
                pMyDev->isRemoteTCSCompleted = true;

                /* If both remote and local TCS transactions complete: */
                if(pMyDev->isLocalTCSCompleted)
                {
                   /* Start opening appropriate logical channels */
                    startOpeningLogicalChannels(pMyDev);
                }
                break;

        /* Received TCSAck from remote 3G-324M endpoint:  */
        case M3GEV_LOCAL_TCS_ACKD:
            {
                /* Assume application defined its device structure: */
                MYDEV * pMyDev;

                /* If both local and remote TCS transactions have completed, can */
                /* initiate the opening of logical channels.                     */

                /* Cache this TCS transaction completion */
                m3g_GetUserInfo(devH, &pMyDev);
                pMyDev->isLocalTCSCompleted = true;

                /* If both remote and local TCS transactions complete: */
                if(pMyDev->isRemoteTCSCompleted)
                {
                   /* Start opening appropriate logical channels */
                    startOpeningLoigicalChannels(pMyDev);
                }
                break;

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
.
.
.
int startOpeningLogicalChannels(MYDEV *pMyDev)
{
   M3G_CAPS_LIST commonCaps;
   /* Retrieve the common H.233 capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->controlDevH, &commonCaps))
   {
      log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }
   /* Configure the H.223 multiplex parameters for audio OLC (not shown)… */
   setOLCH223MuxParameters(pMyDev->h223AudioOLCParams, &commonCaps, AUDIO);

   /* Configure the H.223 multiplex parameters for video OLC (not shown)… */
   setOLCH223MuxParameters(pMyDev-> h223VideoOLCParams, &commonCaps, VIDEO);

   /* Retrieve the common audio capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->audioDevH, &commonCaps))
   {
      log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }
   /* initiate OLC for Tx audio (not shown)… */
```

```
   sendAudioOLC(pMyDev, &commonCaps);

   /* Retrieve the common video capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->videoDevH, &commonCaps))
   {
      log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }
   /* initiate OLC for Tx video (not shown)… */
   sendVideoOLC(pMyDev, &commonCaps);

   return SUCCESS;
} /* End of startOpeningLogicalChannels */
```

See Also

- **m3g_OpenLC( )**
- **m3g_SetTCS( )**
- **m3g_StartH245( )**

m3g_GetParm( )

Name: int m3g_GetParm (deviceHandle, parm)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board or control device handle

M3G_E_PRM_TYPE parm
- parameter type

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_GetParm( )** function retrieves the current value of the specified parameter.

Parameters may be specified for a board device, a control device, or both types of devices. Setting one or more parameters on a board device sets the default values for all control devices associated with that board.

Parameter	Description
deviceHandle	specifies an SRL handle to a board or control device
parm	parameter type specified by the M3G_E_PRM_TYPE enumeration value. For more information, see Table 3, "M3G_PARM_INFO Parameter Types and Parameter Values", on page 217.

This function is only supported in asynchronous mode.

After the function returns M3G_SUCCESS, wait for the M3GEV_GET_PARM_CMPLT event. After the event is returned, use the **sr_getevtdatap( )** Dialogic® Standard Runtime Library function to retrieve the data buffer embedded within the event and cast it as M3G_PARM_INFO to decode the parameter value.

Termination Events

M3GEV_GET_PARM_CMPLT
Indicates specified parameter values were successfully retrieved. The resulting M3G_PARM_INFO structure and its associated parameter value can be retrieved by calling **sr_getevtdatap( )** to retrieve the data buffer embedded within the M3GEV_GET_PARM_CMPLT event and casting it from data type void to data type M3G_PARM_INFO. The data within this buffer must be processed or copied before the next

SRL event is de-queued, at which point this buffer will be de-allocated by the Standard Runtime Library.

M3GEV_GETPARM_FAIL
Indicates that the specified parameter values were not retrieved. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.                */
/*                2) m3g_Open( ) has completed opening the board device handle.       */

int getDefaultH245TerminalType(int boardDevH)
{
   /* retrieve the H.223 multiplex level */
   parameterType = M3G_E_PRM_H245_TERMINAL_TYPE;

   if (M3G_ERROR == m3g_GetParm(boardDevH, parameterType))
   {
      log("Error:  m3g_GetParm(%s)failed - %s\n",
           ATDV_NAMEP(boardDevH), ATDV_ERRMSGP(boardDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of getDefaultH245TerminalType */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }

.
.
.
```

```
void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_GetParm termination: */
        case M3GEV_GET_PARM_EVENTS_CMPLT:
        {
            M3G_PARM_INFO * pParmInfo = (M3G_PARM_INFO *) pSRLEvtDat;
            log("M3GEV_GET_PARM_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            log("Default H.245 terminal type = %d\n",
                pParmInfo->parmValue. h245TerminalType);
            break;
        }

        /* m3g_GetParm Failure indication: */
        case M3GEV_GET_PARM_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_SET_PARM_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_SetParm( )**

m3g_GetUserInfo( )

Name: int m3g_GetUserInfo (deviceHandle, ppUserInfo)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control, audio or video device handle

void ** ppUserInfo
- pointer to void pointer

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: synchronous

Description

The **m3g_GetUserInfo( )** function retrieves a user-defined device handle for an SRL device which was associated with it during **m3g_Open( )** execution.

Parameter	Description
deviceHandle	specifies an SRL handle to a control, audio or video device handle
ppUserInfo	points to a void pointer. Address location returns the void pointer specified in **m3g_Open( )** to be associated with the device. You can use this void pointer as a handle to reference any user-instantiated object, allowing for more rapid device lookup on an SRL device handle.

This function is only supported in synchronous mode. The function returns M3G_SUCCESS if the specified parameter values were successfully returned; otherwise, it returns M3G_ERROR.

Cautions

You must allocate the address location referenced by **ppUserInfo**.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_EnableEvents termination: */
        case M3GEV_OPEN_CMPLT:
        {
            /* Assume user-defined structure: */
            MY_DEVICE_STRUCT * pMyDevStruct;

            log("M3GEV_OPEN_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));

            /* Obtain user-defined handle associated with this SRL handle: */
            m3g_GetUserInfo(devH, &pMyDevStruct);

            /* Mark this device as having completed open processing (not shown)*/
            markDeviceAsOpen(pMyDevStruct);
            break;
        }

       /*
        .
        . Other events not shown…
        .
        */

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_Open( )**

m3g_ModifyMedia( )

Name: int m3g_ModifyMedia (deviceHandle, direction)

Inputs:

SRL_DEVICE_HANDLE deviceHandle	• audio or video device handle
M3G_E_DIRECTION direction	• direction of media flow

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: Data Flow

Mode: asynchronous

Description

The **m3g_ModifyMedia( )** function starts and stops half-duplex media streaming from a specified media device.

Only call this function after the required H.245 forward/reverse logical channels or MONA media preconfigured channels (MPC) have been successfully opened between the local and the remote 3G-324M endpoints.

Parameter	Description
deviceHandle	specifies an SRL handle to an audio or video device
direction	specifies the direction of the media flow. The M3G_E_DIRECTION data type is an enumeration that defines the following values: • M3G_E_IDLE - stop transmission and reception of media from this device • M3G_E_TX - enable transmission of media from this device • M3G_E_RX - enable reception of media from this device • M3G_E_TXRX - enable both transmission and reception of media from this device

This function is only supported in asynchronous mode.

Termination Events

M3GEV_MODIFY_MEDIA_CMPLT
Indicates the specified streaming change was successfully completed. Upon receiving this event, use the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to retrieve the

void data buffer embedded within the event and cast it as a pointer to an M3G_E_DIRECTION enumeration to decode the associated directional change in media.

M3GEV_MODIFY_MEDIA_FAIL
Indicates the specified streaming change failed. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

- It is invalid to call this function with a control device type handle.
- It is invalid to request a directional change of media flow to match the current media direction. If attempted, the function will fail and return a value of M3G_ERROR, and set the associated error code to M3G_E_ERR_INV_STATE.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>

/* Preconditions:                                          */
/* 1) 3G-324M Library Session has already been started.    */
/* 2) m3g_Open( ) has completed opening the control and    */
/*     media devices.                                      */
/* 3) Only the H.245 reverse unidirectional logical channel */
/*     for this audio device has been opened successfully.  */
/*     The forward logical channel has not yet been opened. */

int startAudioRcvOnly(int audioDevH)
{
   if (M3G_ERROR == m3g_ModifyMedia(audioDevH, M3G_E_RX))
   {
      log("Error: m3g_ModifyMedia (%s)failed - %s\n",
      ATDV_NAMEP(audioDevH), ATDV_ERRMSGP(audioDevH));
      /* handle error */
   }
   return SUCCESS;
} /* End of startAudioRcvOnly */


/* Preconditions:                                          */
/* 1) 3G-324M Library Session has already been started.    */
/* 2) m3g_Open( ) has completed opening the control and    */
/*     media devices.                                      */
/* 3) Only the H.245 forward unidirectional logical channel */
/*     for this audio device has been opened successfully.  */
/*     The reverse logical channel has not yet been opened. */

int startAudioXmtOnly(int audioDevH)
{
```

```
   if (M3G_ERROR == m3g_ModifyMedia(audioDevH, M3G_E_TX))
   {
      log("Error: m3g_ModifyMedia (%s)failed - %s\n",
      ATDV_NAMEP(audioDevH), ATDV_ERRMSGP(audioDevH));
      /* handle error */
   }
   return SUCCESS;
} /* End of startAudioXmtOnly */


/* Preconditions:                                          */
/* 1) 3G-324M Library Session has already been started.    */
/* 2) m3g_Open( ) has completed opening the control and    */
/*      media devices.                                     */
/* 3) Both the H.245 forward and reverse unidirectional    */
/*      logical channels for this audio device have been   */
/*      opened successfully.                               */

int startAudioXmtAndRcv(int audioDevH)
{
   if (M3G_ERROR == m3g_ModifyMedia(audioDevH, M3G_E_TXRX))
   {
      log("Error: m3g_ModifyMedia (%s)failed - %s\n",
      ATDV_NAMEP(audioDevH), ATDV_ERRMSGP(audioDevH));
      /* handle error */
   }
   return SUCCESS;
} /* End of startAudioXmtAndRcv */


/* Preconditions:                                          */
/* 1) 3G-324M Library Session has already been started.    */
/* 2) m3g_Open( ) has completed opening the control and    */
/*      media devices.                                     */
/* 3) Both the H.245 forward and reverse unidirectional    */
/*      logical channels for this audio device have been   */
/*      closed.                                            */

int stopAudioXmtAndRcv(int audioDevH)
{
   if (M3G_ERROR == m3g_ModifyMedia(audioDevH, M3G_E_IDLE))
   {
      log("Error: m3g_ModifyMedia (%s)failed - %s\n",
      ATDV_NAMEP(audioDevH), ATDV_ERRMSGP(audioDevH));
      /* handle error */
   }
   return SUCCESS;
} /* End of stopAudioXmtAndRcv */
.
.
.
.
.
/* within SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
      process_event();
   }
.
.
.
void process_event(void)
{
   /* process the SRL events */
   int evType = sr_getevttype();
   int devH = sr_getevtdev();
```

```
    void *pSRLEvtData = sr_getevtdatap();
    switch(evType)
    {
       /*
       .
       . Other events not shown
       .
       */
       /* Successful m3g_ModifyMedia termination: */
       case M3GEV_MODIFY_MEDIA_CMPLT:
       {
            void *pEvtData = sr_getevtdatap();
            M3G_E_DIRECTION direction = *((M3G_E_DIRECTION*)pEvtData);
            log("M3GEV_MODIFY_MEDIA_CMPLT(dir=%d) for device = %s\n",
                direction, ATDV_NAMEP(devH));
            break;
        }

       /* m3g_ ModifyMedia Failure indication: */
       case M3GEV_MODIFY_MEDIA_FAIL:
       {
          M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
          log("ERROR: M3GEV_MODIFY_MEDIA_FAIL for device = %s\n",
          ATDV_NAMEP(devH));
          log(" Error value = %d\n",(int)*pError);
          /* handle error…*/
          break;
       }

       default:
          printf("Received unknown event = %d for device = %s\n",
          evType, ATDV_NAMEP(devH));
          break;
       }
}
```

See Also

- **m3g_StartMedia( )**

m3g_Open( )

Name: int m3g_Open (deviceName, pOpenInfo, pUserInfo)

Inputs:

const char *deviceName	• pointer to device name
M3G_OPEN_INFO *pOpenInfo	• reserved for future use
void *pUserInfo	• pointer to user-defined device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_Open( )** function opens the specified device and returns a unique device handle to identify the device. All subsequent references to this device must be made using this handle until the device is closed.

Parameter	Description
deviceName	points to a character string defining the type, the board, and the channel of the device to be opened. Valid formats include: • m3gB*m*T*n* – control device where *m* is the board number and *n* is the channel number • m3gB*m* – board device where *m* is the board number • m3gB*m*T*n*:AUDIO*p* – audio device of instance number *p* to be multiplexed/demultiplexed in aggregate of control device m3gB*m*T*n* • m3gB*m*T*n*:VIDEO*p* – video device of instance number *p* to be multiplexed/demultiplexed in aggregate of control device m3gB*m*T*n* ***Note:*** Board number *m* must be 1 in all device names (board, control, audio, and video). For more information on the types of devices, see Device Types section.
pOpenInfo	Reserved for future use and currently ignored.
pUserInfo	points to a user-defined device handle to associate with the SRL device handle. To retrieve this handle, use **m3g_GetUserInfo( )**.

This function is only supported in asynchronous mode.

If the function is called with valid arguments, a valid device handle is returned immediately. Before using this device handle in other function calls, you must wait for the

M3GEV_OPEN_CMPLT event indicating that the device resource allocation and instantiation process has completed.

If the function is called and M3GEV_OPEN_FAIL is returned, a device handle is also returned. You must call **m3g_Close( )** to de-allocate the handle returned by **m3g_Open( )**.

Device Types

Each 3G-324M endpoint is a composite or aggregate of several device types:

board device

The board device is used to set global default values. The board device handle is used in **m3g_SetParm( )** function calls to specify parameter values for all applicable control, audio, and video device instances subsequently opened on the specified board. It is used in **m3g_EnableEvents( )** function calls to enable unsolicited events. It is also used by other functions such as **m3g_SetVendorId( )** and **m3g_StartTrace( )**.

The board device name is "m3gB*m*", where "*m*" is the specified board number.

Note: Only one board, m3gB1, is currently supported.

Note: Any function called on a board device affects all instances on that board device. For example, if an application resets the board device via **m3g_Reset( )**, all channels are reset on that board device. Similarly, if an application uses the **m3g_SetVendorId( )** and **m3g_StartTrace( )** functions on the board device level, all channels on the board device are affected by these functions. Be aware of this behavior, in particular if two applications access the same board device.

control device

The control device is the primary handle for 3G-324M endpoint control. It is used to manage the following functional interfaces:

- H.245 control – provides H.245 control operations for a given 3G-324M endpoint. This device is automatically associated with the aggregate H.223 multiplex/demultiplex as logical channel 0 when the device is opened.
- H.223 multiplex/demultiplex – permits physical connections to and from the H.223 multiplex/demultiplex over CT Bus timeslots or over IP using an Nb User Plane (Nb UP) protocol. CT Bus timeslots may be used to connect to appropriate T1/E1 bearer channels which transport the aggregate data off-board to route the H.223 multiplex/demultiplex to other 3G-324M endpoints. The Nb UP may be used to route bearer control and transport of the H.223 multiplex/demultiplex within the 3G core network Release 4 and later.

Connections and disconnections between the H.223 multiplex/demultiplex aggregate are made using device management API functions. If the aggregate is routed over a DS0 timeslot on the CT Bus, **dev_Connect( )** and **dev_Disconnect( )** are used. If the aggregate is routed over the Nb UP, **dev_PortConnect( )** and **dev_PortDisconnect( )** are used.

The control device name is "m3gB*m*T*n*", where "*m*" is the specified board number and "*n*" is the specified channel number.

audio device

The audio device represents the audio connection to and from the H.223 multiplex. This device type does not initiate or terminate audio streams. The audio device connects another R4 device type, such as an IP media device (ipmBxCy) or a multimedia device (mmBxCy), which

provides the source and destination for the associated audio data streams, through the **dev_PortConnect( )** and **dev_PortDisconnect( )** functions.

Similarly, the audio device may be connected to a digital network interface device (dtiBxTy) or voice device (dxxxBxCy) through the **dev_Connect( )** and **dev_Disconnect( )** functions.

Prior to multiplexing/demultiplexing, each audio device must establish a connection to an R4 audio device using **dev_PortConnect( )**.

The audio device name is "m3gB*m*T*n*:AUDIO*p*", where "m3gB*m*T*n*" is the specified control device into and out of which the audio device should be multiplexed/demultiplexed; "*p*" in "AUDIO*p*" represents the number of the audio instance and is used to differentiate multiple audio devices which may comprise an H.223 aggregate.

Note: Only one audio streaming connection to and from the H.223 aggregate is currently supported.

video device

This device represents the video connection to and from the H.223 multiplex. This device type does not initiate or terminate video streams. The video device connects another R4 device type, such as an IP media device (ipmBxCy) or a multimedia device (mmBxCy), which provides the source and destination for the associated video data streams, through the **dev_PortConnect( )** and **dev_PortDisconnect( )** functions.

Prior to multiplexing/demultiplexing, each video device must establish a connection to an R4 video device using **dev_PortConnect( )**.

The video device name is "m3gB*m*T*n*:VIDEO*p*", where "m3gB*m*T*n*" is the specified control device into and out of which the video device should be multiplexed/demultiplexed; "*p*" in "VIDEO*p*" represents the number of the video instance and is used to differentiate multiple video devices which may comprise an H.223 aggregate.

Note: Only one video streaming connection to and from the H.223 aggregate is currently supported.

Termination Events

M3GEV_OPEN_CMPLT

indicates the specified device type successfully opened

M3GEV_OPEN_FAIL

indicates the specified device type failed to open successfully. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

Only one device type can be opened at a time by **m3g_Open( )**. Device name strings cannot be concatenated to open one composite 3G-324M endpoint device.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) Assume all 3G-324M devices are on a single board, m3gB1 */
/*                3) Assume global device table is defined elsewhere         */
int OpenDevs(int numDevices)
{
   char devName[80];
   int devNum;

   /* Open board device to set device default settings */
   boardDev = m3g_Open("m3gB1", NULL);
   if (0 > boardDev)
   {
      log("Error:  m3g_Open(m3gB1,NULL) failed - ERR = %d\n",
           boardDev);
      /* handle error… */
   }
   /* Use boardDev in m3g_SetParm( ) to configure board default settings (not shown…) */

   for(devNum=1; devNum<=numDevices; devNum++)
   {
      /* Open control device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d", devNum);
      devTbl[devNum].cDev = m3g_Open(devName, NULL, &devTbl[devNum]);
      if (0 > devTbl[devNum].cDev)
      {
         log("Error:  m3g_Open(%s)failed\n", devName);
         /* handle error… */
      }

      /* Open audio device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:AUDIO1", devNum);
      devTbl[devNum].aDev = m3g_Open(devName, NULL, &devTbl[devNum]);
      if (0 > devTbl[devNum].aDev)
      {
         log("Error:  m3g_Open(%s)failed\n", devName);
         /* handle error… */
      }

      /* Open video device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:VIDEO1", devNum);
      devTbl[devNum].vDev = m3g_Open(devName, NULL, &devTbl[devNum]);
      if (0 > devTbl[devNum].vDev)
      {
         log("Error:  m3g_Open(%s)failed\n", devName);
         /* handle error… */
      }
   }
   return SUCCESS;
} /* End of OpenDevs */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
```

```
            process_event();
    }
.
.
.
void process_event(void)
{
    /* process the SRL events */
    int evType        = sr_getevttype();
    int devH          = sr_getevtdev();
    void *pSRLEvtData = sr_getevtdatap();

    switch(evType)
    {
        /*
         .
         . Other events not shown…
         .
         */
         /* Successful m3g_Open termination: */
         case M3GEV_OPEN_CMPLT:
             log("M3GEV_OPEN_CMPLT for device = %s\n",
                   ATDV_NAMEP(devH));
             /* Device is ready for 3G-324M processing */
             deviceIsReady(devH);  /* proceed with 3G-324M processing (not shown)*/
             break;

         /* m3g_Open Failure indication: */
         case M3GEV_OPEN_FAIL:
         {
             M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
             log("ERROR:  Device = %s received M3GEV_OPEN_FAIL\n",
                   ATDV_NAMEP(devH));
             log("        Error  = %d\n", *pError);

             /* handle error…*/
             break;
         }
         default:
            printf("Received unknown event = %d for device = %s\n",
                    evType, ATDV_NAMEP(devH));
            break;
    }
}
```

See Also

- **m3g_Close( )**

m3g_OpenEx( )

Name: int m3g_OpenEx (deviceName, pOpenInfo, pUserInfo, mode)

Inputs:

const char *deviceName	• pointer to device name
M3G_OPEN_INFO *pOpenInfo	• reserved for future use
void *pUserInfo	• pointer to user-defined device handle
unsigned short mode	• mode of operation

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: asynchronous or synchronous

Description

The **m3g_OpenEx( )** function opens the specified device and returns a unique device handle to identify the device. All subsequent references to this device must be made using this handle until the device is closed.

The **m3g_OpenEx( )** function allows you to choose synchronous or asynchronous mode. If you require operation in synchronous mode, use **m3g_OpenEx( )** instead of **m3g_Open( )**.

Parameter	Description
deviceName	points to a character string defining the type, the board, and the channel of the device to be opened. Valid formats include: • m3gB*m*T*n* – control device where *m* is the board number and *n* is the channel number • m3gB*m* – board device where *m* is the board number • m3gB*m*T*n*:AUDIO*p* – audio device of instance number *p* to be multiplexed/demultiplexed in aggregate of control device m3gB*m*T*n* • m3gB*m*T*n*:VIDEO*p* – video device of instance number *p* to be multiplexed/demultiplexed in aggregate of control device m3gB*m*T*n* ***Note:*** Board number *m* must be 1 in all device names (board, control, audio, and video). For more information on the types of devices, see Device Types section in the **m3g_Open( )** function description.
pOpenInfo	Reserved for future use and currently ignored.

Parameter	Description
pUserInfo	points to a user-defined device handle to associate with the SRL device. To retrieve this handle, use **m3g_GetUserInfo( )**.
mode	specifies mode of operation. Valid values are: • EV_ASYNC – asynchronous mode • EV_SYNC – synchronous mode There is no default setting for mode.

In synchronous mode, if the function is successful, a valid device handle is returned and can be used for further processing.

In asynchronous mode, if the function is called with valid arguments, a valid device handle is returned immediately. Before using this device handle in other function calls, you must wait for the M3GEV_OPEN_CMPLT event indicating that the device resource allocation and instantiation process has completed.

In asynchronous mode, if the function is called and M3GEV_OPEN_FAIL event is returned, a device handle is also returned. You must call **m3g_Close( )** to de-allocate the handle returned by **m3g_OpenEx( )**.

Termination Events

M3GEV_OPEN_CMPLT
: Indicates that the specified device type successfully opened.

M3GEV_OPEN_FAIL
: Indicates that the specified device type failed to open successfully. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

Only one device type can be opened at a time by **m3g_OpenEx( )**. Device name strings cannot be concatenated to open one composite 3G-324M endpoint device.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Synchronous Example

```
/* Synchronous m3g_OpenEx( ) sample code */

/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
```

```
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) Assume all 3G-324M devices are on a single board, m3gB1 */
/*                3) Assume global device table is defined elsewhere         */
int OpenDevs(int numDevices)
{
   char devName[80];
   int devNum;

   /* Open board device to set device default settings */
   boardDev = m3g_OpenEx("m3gB1", NULL, NULL, EV_SYNC);
   if (0 > boardDev)
   {
      log("Error:  m3g_OpenEx(m3gB1,NULL) failed - ERR = %d\n",
           boardDev);
      /* handle error… */
   }
   /* Use boardDev in m3g_SetParm( ) to configure board default settings (not shown ) */

   for(devNum=1; devNum<=numDevices; devNum++)
   {
      /* Open control device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d", devNum);
      devTbl[devNum].cDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_SYNC);
      if (0 > devTbl[devNum].cDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }

      /* Open audio device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:AUDIO1", devNum);
      devTbl[devNum].aDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_SYNC);
      if (0 > devTbl[devNum].aDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }

      /* Open video device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:VIDEO1", devNum);
      devTbl[devNum].vDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_SYNC);
      if (0 > devTbl[devNum].vDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }
   }

   /*
    * All 3G device are immediately ready for 3G-324M processing. No open termination
    * events are queued.
    */
   return SUCCESS;
} /* End of OpenDevs */
```

Asynchronous Example

```
/* Asynchronous m3g_OpenEx( ) sample code */

/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
```

```
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) Assume all 3G-324M devices are on a single board, m3gB1 */
/*                3) Assume global device table is defined elsewhere         */
int OpenDevs(int numDevices)
{
   char devName[80];
   int devNum;

   /* Open board device to set device default settings */
   boardDev = m3g_OpenEx("m3gB1", NULL, NULL, EV_ASYNC);
   if (0 > boardDev)
   {
      log("Error:  m3g_OpenEx(m3gB1,NULL) failed - ERR = %d\n",
           boardDev);
      /* handle error… */
   }
   /* Use boardDev in m3g_SetParm( ) to configure board default settings (not shown ) */

   for(devNum=1; devNum<=numDevices; devNum++)
   {
      /* Open control device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d", devNum);
      devTbl[devNum].cDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_ASYNC);
      if (0 > devTbl[devNum].cDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }

      /* Open audio device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:AUDIO1", devNum);
      devTbl[devNum].aDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_ASYNC);
      if (0 > devTbl[devNum].aDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }

      /* Open video device for 3G-324M endpoint */
      sprintf(devName, "m3gB1T%d:VIDEO1", devNum);
      devTbl[devNum].vDev = m3g_OpenEx(devName, NULL, &devTbl[devNum], EV_ASYNC);
      if (0 > devTbl[devNum].vDev)
      {
         log("Error:  m3g_OpenEx(%s)failed\n", devName);
         /* handle error */
      }
   }
   return SUCCESS;
} /* End of OpenDevs */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.
void process_event(void)
```

```
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown
        .
        */
        /* Successful m3g_OpenEx termination: */
        case M3GEV_OPEN_CMPLT:
            log("M3GEV_OPEN_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            /* Device is ready for 3G-324M processing */
            deviceIsReady(devH);  /* proceed with 3G-324M processing (not shown)*/
            break;

        /* m3g_OpenEx Failure indication: */
        case M3GEV_OPEN_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  Device = %s received M3GEV_OPEN_FAIL\n",
                  ATDV_NAMEP(devH));
            log("        Error  = %d\n", *pError);

            /* handle error */
            break;
        }
        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_Close( )**
- **m3g_Open( )**

m3g_OpenLC( )

Name: int m3g_OpenLC (deviceHandle, pH223LCParameters, capabilityType, pMediaCapability)

Inputs:

SRL_DEVICE_HANDLE deviceHandle	• control device handle
M3G_H223_LC_PARAMS *pH223LCParameters	• pointer to M3G_H223_LC_PARAMS structure
M3G_E_CAPABILITY capabilityType	• type of capability specified in M3G_CAPABILITY union
M3G_CAPABILITY *pMediaCapability	• pointer to M3G_CAPABILITY union

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_OpenLC( )** function sends an OpenLogicalChannel request to multiplex a media channel in H.223 from the local 3G-324M endpoint to the remote 3G-324M endpoint using the specified adaptation layer format, interleaving the specified audio or video capability media format.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
pH223LCParameters	points to the M3G_H223_LC_PARAMS structure specifying the H223LogicalChannelParameters elements to be encoded within the OpenLogicalChannel message
capabilityType	indicates the type of capability specified in the M3G_CAPABILITY union, referenced by **pMediaCapability**. The data type is an enumeration that defines the following values: • M3G_E_AUDIO_CAPABILITY – audio capability • M3G_E_VIDEO_CAPABILITY – video capability
pMediaCapability	points to the M3G_CAPABILITY union which must only be of data type M3G_AUDIO_CAPABILITY or M3G_VIDEO_CAPABILITY

Call this function after invoking **m3g_GetMatchedCaps( )** for compatible H.223 abstraction layer capabilities to initialize the M3G_H223_LC_PARAMS structure, and either an audio or video capability to initialize the M3G_CAPABILITY union with a transmit media capability obtained

from the intersection of common capabilities between the local and remote 3G-324M endpoint as established from TerminalCapabilitySet exchange. The element settings of this structure and union should be a transmit audio or video capability element copied from the results of **m3g_GetMatchedCaps( )**. Modifying settings of a matching capability returned from **m3g_GetMatchedCaps( )** may lead to undefined behavior.

When an OpenLogicalChannel request is received from the remote 3G-324M endpoint, an M3GEV_REMOTE_OLC_RCVD event is queued to the application which includes information about the requested H.223 abstraction layer and media formats. This logical channel request may be positively acknowledged or rejected by calling **m3g_RespondToOLC( )**.

The **m3g_OpenLC( )** function completes only after receiving an acknowledgement from the remote 3G-324M endpoint. The function may receive an OpenLogicalChannelAck response from the remote 3G-324M endpoint, indicated by the M3GEV_OPEN_LC_CMPLT event. The function may receive an OpenLogicalChannelReject response from the remote 3G-324M endpoint, indicated by the M3GEV_OPEN_LC_FAIL event.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_OPEN_LC_CMPLT
: Indicates an OpenLogicalChannelAck response has been successfully received from the remote 3G-324M endpoint, acknowledging the OpenLogicalChannel request issued previously on this control device. Upon receiving this event, use the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to retrieve the void data buffer embedded within the event and cast it as a pointer to an M3G_REMOTE_OLCACK_RESP structure to decode the assigned logical channel number and its forward direction media capability.

M3GEV_OPEN_LC_FAIL
: Indicates the specified capability in the OpenLogicalChannel request was not positively acknowledged from the remote 3G-324M endpoint. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

- It is invalid to call this function with a board, audio or video device type handle.
- No attempt at opening logical channels may be initiated until MasterSlaveDetermination and TerminalCapabilitySet transactions, both the request and the response, have completed in each direction with the remote 3G-324M endpoint.
- The specified media direction must be set to M3G_E_TX in the M3G_AUDIO_CAPABILITY structure, as only forward, unidirectional logical channels may be opened.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the                   */
/*                   applicable control, audio and or video devices          */
/*                   associated with the 3G-324M bearer channel.             */
/*                3) The control, audio, and or video devices have all been  */
/*                   interconnected to their respective network and ipm or   */
/*                   mm devices using the dev_PortConnect( ) or              */
/*                   dev_Connect( )functions.                                */
/*                4) The default simultaneous caps table has been set using  */
/*                   the m3g_GetLocalCap( ) and m3g_SetTCS( )                */
/*                   function (not shown)                                    */
.
.
.
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Received TCS from remote 3G-324M endpoint:  */
        case M3GEV_REMOTE_TCS_RCVD:
        {
           /* Assume application defined its device structure: */
            MYDEV * pMyDev;
            log("M3GEV_REMOTE_TCS_RCVD for device = %s\n",
            ATDV_NAMEP(devH));

           /* If both local and remote TCS transactions have completed, can */
           /* initiate the opening of logical channels if necessary. */

           /* Cache this TCS transaction completion */
           m3g_GetUserInfo(devH, &pMyDev);
           pMyDev->isRemoteTCSCompleted = true;

           /* If both remote and local TCS transactions complete: */
           if(pMyDev->isLocalTCSCompleted)
```

```
     {
        /* Open any transmit media channels that may not have */
        /* been established via MONA MPC procedures           */
        if(pMyDev->audioMPCEstablished == false)
        {
            openAudioLogicalChannel(pMyDev);
        }
        if(pMyDev->videoMPCEstablished == false)
        {
            openVideoLogicalChannel(pMyDev);
        }
   }
   break;
}

/* Received TCSAck from remote 3G-324M endpoint: */
case M3GEV_LOCAL_TCS_ACKD:
{
   /* Assume application defined its device structure: */
   MYDEV * pMyDev;
   log("M3GEV_REMOTE_TCS_ACKD for device = %s\n",
   ATDV_NAMEP(devH));

   /* If both local and remote TCS transactions have completed, can */
   /* initiate the opening of logical channels. */
   /* Cache this TCS transaction completion */
   m3g_GetUserInfo(devH, &pMyDev);
   pMyDev->isLocalTCSCompleted = true;

   /* If both remote and local TCS transactions complete: */
   if(pMyDev->isRemoteTCSCompleted)
   {
      /* Open any transmit media channels that may not have */
      /* been established via MONA MPC procedures           */
      if(pMyDev->audioMPCEstablished == false)
      {
         openAudioLogicalChannel(pMyDev);
      }
      if(pMyDev->videoMPCEstablished == false)
      {
         openVideoLogicalChannel(pMyDev);
      }
   }
   break;
}

case M3GEV_SEND_MONA_PREF_MSG:
case M3GEV_MONA_PREF_MSG_RCVD:
{
   M3G_MONA_TXRX_MPC_SUPPORT* pMPC = (M3G_MONA_TXRX_MPC_SUPPORT*)pSRLEvtData;
   log("MONA Pref_Msg %s, rxMPC:0x%x txMPC:0x%x\n",
       (evType == M3GEV_SEND_MONA_PREF_MSG) ? "sent" : "rcvd",
       pMPC->rxMPCMask, pMPC->txMPCMask);
   break;
}

case M3GEV_TX_MPC_ESTABLISHED:
case M3GEV_RX_MPC_ESTABLISHED:
{
   M3G_MONA_MPC* pMPC = (M3G_MONA_MPC*)pSRLEvtData;
   M3G_LOGICAL_CHANNEL_NUMBER lcn = pMPC->logicalChannelNumber;
   M3G_E_DIRECTION direction = (M3GEV_TX_MPC_ESTABLISHED==evtType) ? M3G_E_TX : M3G_E_RX;
   M3G_E_CAPABILITY mediaType = pMPC->capabilityType;

   /* Assume application defined its device structure: */
   MYDEV * pMyDev;
   m3g_GetUserInfo(devH, &pMyDev);
```

```
   log("MPC established: %s %s LCN:%d\n",
       (M3G_E_TX == direction) ? "TX" : "RX",
       (M3G_E_AUDIO_CAPABILITY == mediaType) ? "AUDIO" : "VIDEO",
       lcn);

   if (M3G_E_AUDIO_CAPABILITY == pMPC->capabilityType)
   {
      /* Cache that this TX audio MPC is established so it need not be opened */
      /* via legacy H.245 openLogicalChannel procedures via m3g_OpenLC( )     */
      if (M3G_E_TX == direction)
      {
         pMyDev->audioMPCEstablished = true;
      }
      /* Activate audio streaming in specified direction */
      ActivateAudioMedia(direction);
   }
   else
   {

      /* Cache that this TX video MPC is established so it need not be opened */
      /* via legacy H.245 openLogicalChannel procedures via m3g_OpenLC( )     */
      if (M3G_E_TX == direction)
      {
         pMyDev->videoMPCEstablished = true;

      }
      /* Activate video streaming in specified direction */
      ActivateVideoMedia(direction);
   }
   break;
}

/* Received OLCAck from remote 3G-324M endpoint: */
case M3GEV_OPEN_LC_CMPLT:
{
   /* Assume application defined its device structure: */
   MYDEV * pMyDev;
   M3G_REMOTE_OLCACK_RESP* pOLCAckResp =
     (M3G_REMOTE_OLCACK_RESP *) pSRLEvtData;

   /* Cache this TCS transaction completion */
   m3g_GetUserInfo(devH, &pMyDev);

   /* Must determine if this was for our audio or video OLC (not shown): */
   if (true == isCapTypeAudio(&pOLCAckResp->mediaCapability))
   {
      pMyDev->isAudioFwdOLCAcked = true;
      pMyDev->fwdAudioLCN = pOLCAckResp->logicalChannelNumber;

      /* Activate audio streaming in transmit direction (not shown)*/
      ActivateAudioMedia(M3G_E_TX);
   }
   else /* else video: */
   {
      pMyDev->isVideoFwdOLCAcked = true;
      pMyDev->fwdVideoLCN = pOLCAckResp->logicalChannelNumber;

       /* Activate video streaming in transmit direction (not shown)*/
       ActivateVideoMedia(M3G_E_TX);
   }
   break;
}

/* m3g_OpenLC failure indication - perhaps received OLCReject or other failure: */
case M3GEV_OPEN_LC_FAIL:
{
   M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
```

```
            log("ERROR: M3GEV_OPEN_LC_FAIL for device = %s\n",
                ATDV_NAMEP(devH));
            log(" Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
         }

         default:
            printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
            break;
      }
   }
.
.
.

int openAudioLogicalChannel(MYDEV *pMyDev)
{
   M3G_CAPS_LIST h223CommonCaps;
   M3G_CAPS_LIST audioCommonCaps;
   int index;

   /* Retrieve the common H.233 capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->controlDevH, &h223CommonCaps))
   {
      log("Error: m3g_GetMatchedCaps(%s)failed - %s\n",
      ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));

      /* handle error… */
   }

   /* Retrieve the common audio capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->audioDevH, &audioCommonCaps))
   {
      log("Error: m3g_GetMatchedCaps(%s)failed - %s\n",
      ATDV_NAMEP(pMyDev->audioDevH), ATDV_ERRMSGP(pMyDev->audioDevH));

      /* handle error… */
   }
   /* Choose the MASTER's most preferred audio capability from among the common types */
   /* This should be the first element in the matched capability list. */

   for(index = 0;

       ((index < audioCommonCaps.numCaps) && (index < MAX_CAPABILITIES_PER_DEVICE));
       index++)
   {
      /* Any additional capability selection criteria and logic not shown */
      if (true == isAudioPreferred(&audioCommonCaps.capability[index]))
      {
         if (M3G_ERROR == m3g_OpenLC(pMyDev->controlDevH,
                                     pMyDev->h223AudioOLCParams,
                                     M3G_E_AUDIO_CAPABILITY,
                                     &audioCommonCaps.capability[index]))
         {

            log("Error: m3g_OpenLC(%s)failed - %s\n",
            ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));

            /* handle error… */
         }
         break;
      }
   } /* endFor */
} /* End of openAudioLogicalChannel */
```

```
int openVideoLogicalChannel(MYDEV *pMyDev)
{
   M3G_CAPS_LIST h223CommonCaps;
   M3G_CAPS_LIST videoCommonCaps;
   int index;

   /* Retrieve the common H.233 capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->controlDevH, &h223CommonCaps))
   {
      log("Error: m3g_GetMatchedCaps(%s)failed - %s\n",
      ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   /* Retrieve the common video capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->videoDevH, &videoCommonCaps))
   {
      log("Error: m3g_GetMatchedCaps(%s)failed - %s\n",
      ATDV_NAMEP(pMyDev->videoDevH), ATDV_ERRMSGP(pMyDev->videoDevH));
      /* handle error… */
   }

   /* Choose the MASTER's most preferred video capability from among the common types */
   /* This should be the first element in the matched capability list. */
   for(index = 0;
       ((index < videoCommonCaps.numCaps) && (index < MAX_CAPABILITIES_PER_DEVICE));
       index++)
   {

      /* Any additional capability selection criteria and logic not shown */
      if (true == isVideoPreferred(&videoCommonCaps.capability[index]))
      {
         if (M3G_ERROR == m3g_OpenLC(pMyDev->controlDevH,
                                     pMyDev->h223VideoOLCParams,
                                     M3G_E_VIDEO_CAPABILITY,
                                     &videoCommonCaps.capability[index]))
         {
            log("Error: m3g_OpenLC(%s)failed - %s\n",
            ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
            /* handle error… */
         }
         break;
      }
   } /* endFor */

} /* End of openVideoLogicalChannel */
```

See Also

- **m3g_CloseLC( )**
- **m3g_GetMatchedCaps( )**
- **m3g_RespondToOLC( )**

m3g_Reset( )

Name: int m3g_Reset(deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board device or control device handle

Returns: M3G_SUCCESS for success
M3G_ERROR for failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_Reset( )** function resets all devices that may have been opened and not closed by a previous process for the specified board device or control device, including associated audio and video devices. The devices are reset to their initial state. The state of m3g devices includes interconnections, capability settings, and streaming. It does not include parameter settings, as parameters set via **m3g_SetParm( )** are persistent through calling **m3g_Reset( )**.

A common use of this function is to call it at the beginning of an application in order to make sure that the devices are properly reset. This function can also be called upon indication of an unrecoverable error.

The function returns the M3GEV_RESET_COMPLT event if it successfully recovered one or more devices, or if there were no devices to recover.

Parameter	Description
deviceHandle	specifies an SRL handle to a board device or control device

Termination Events

M3GEV_RESET_COMPLT
Indicates that the specified devices were successfully reset.

M3GEV_RESET_FAIL
Indicates that the specified devices failed to be reset. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

- This function disconnects all m3g device interconnections. Any previously configured port or timeslot connections must be re-initialized by calling **dev_PortConnect( )** or **dev_Connect( )** after calling **m3g_Reset( )**.

- This function clears all default local terminal capabilities. Any previously specified capabilities must be re-initialized by calling **m3g_SetTCS( )** after calling **m3g_Reset( )**.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.   */
/*                2) All 3G-324M devices have been opened.               */
int HandleError(SRL_DEVICE_HANDLE devH, int evType)
{
   M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
   log("ERROR:  Device = %s received failure event: %d\n",
       ATDV_NAMEP(devH), evType);
   log("        Error  = %d\n", *pError);

   if (M3G_ERROR == m3g_Reset(devH))
   {
      log("Error:  m3g_Reset(%s) failed - %s\n",
          ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error */
   }

   return SUCCESS;
} /* End of OpenDevs */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.
void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
```

```
        . Other events not shown
        .
        */

        /* For all device failure indications (excluding M3GEV_OPEN_FAIL): */
        case M3GEV_ENABLE_EVENTS_FAIL:
        case M3GEV_DISABLE_EVENTS_FAIL:
        case M3GEV_SET_PARM_FAIL:
        case M3GEV_GET_PARM_FAIL:
        case M3GEV_START_H245_FAIL:
        case M3GEV_MSD_FAILED:
        case M3GEV_STOP_H245_FAIL:
        case M3GEV_GET_LOCAL_CAPS_FAIL:
        case M3GEV_SET_TCS_FAIL:
        case M3GEV_OPEN_LC_FAIL:
        case M3GEV_RESPOND_TO_LC_FAIL:
        case M3GEV_CLOSE_LC_FAIL:
        case M3GEV_START_MEDIA_FAIL:
        case M3GEV_MODIFY_MEDIA_FAIL:
        case M3GEV_STOP_MEDIA_FAIL:
        case M3GEV_SEND_H245_UII_FAIL:
        case M3GEV_SEND_H245_MISC_CMD_FAIL:
            HandleError(devH, evType);
            break;
        }

        case M3GEV_RESET_CMPLT:
           /* Set 3G device to initial state as defined within user application code (
                  not shown): */
           ReinitalizeState(devH);
           break;

        case M3GEV_RESET_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_RESET_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /*
             * No further course of action for m3g_Reset failures.
               Simply exit at this point...
             */
            exit(1);
            break;
        }


        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
    }
```

See Also

None.

m3g_RespondToOLC( )

Name:	int m3g_RespondToOLC (deviceHandle, lcn, olcResponse)	
Inputs:	SRL_DEVICE_HANDLE deviceHandle	• control device handle
	M3G_LOGICAL_CHANNEL_NUMBER lcn	• logical channel number of incoming request
	M3G_E_OLC_RESP_TYPE olcResponse	• M3G_E_OLC_RESP_TYPE response to incoming OpenLogicalChannel request
Returns:	M3G_SUCCESS if successful M3G_ERROR on failure	
Includes:	srllib.h m3glib.h m3gevts.h m3gerrs.h	
Category:	H.245 Control	
Mode:	asynchronous	

Description

The **m3g_RespondToOLC**( ) function sends a response to an incoming OpenLogicalChannel request from the remote 3G-324M endpoint. The response may be either OpenLogicalChannelAck or OpenChannelReject, as specified in the **olcResponse** parameter.

Call this function after receiving the M3GEV_REMOTE_OLC_RCVD event, which indicates that an incoming OpenLogicalChannel request was received from the remote 3G-324M endpoint.

The M3GEV_REMOTE_OLC_RCVD event includes information about the requested H.223 adaptation layer and media formats. Upon receiving this event, use the Dialogic® Standard Runtime Library function **sr_getevtdatap**( ) to retrieve the void data buffer embedded within the event and cast it as a pointer to an M3G_REMOTE_OLC_REQ structure.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device

Parameter	Description
lcn	specifies the logical channel number of the incoming OpenLogicalChannel request
olcResponse	indicates whether to positively acknowledge the request through the OpenLogicalChannelAck message or to reject the request through the OpenLogicalChannelReject message. Cause code for a rejection is also provided. The data type is an enumeration that defines the following values: • M3G_E_OLCACK – OLCAck response • M3G_E_OLCREJ_UNSPECIFIED – OLCReject cause unspecified • M3G_E_OLCREJ_UNS_REV_PARM – OLCReject cause unsuitableReverseParameters • M3G_E_OLCREJ_DATATYP_NOT_SUP – OLCReject cause dataTypeNotSupported • M3G_E_OLCREJ_DATATYP_NOT_AVAIL – OLCReject cause dataTypeNotAvailable • M3G_E_OLCREJ_DATATYP_AL_UNSUP – OLCReject cause dataTypeALCombintationNotSupported • M3G_E_OLCREJ_MC_CHAN_NOT_ALWD – OLCReject cause multicatChannelNotAllowed • M3G_E_OLCREJ_INSUFF_BW – OLCReject cause insufficientBandwidth • M3G_E_OLCREJ_STACK_FAILED – OLCReject cause separateStackEstablishmentFailed • M3G_E_OLCREJ_INV_SESSIONID – OLCReject cause invalidSessionID • M3G_E_OLCREJ_MS_CONFLICT – OLCReject cause masterSlaveConflict • M3G_E_OLCREJ_WAIT_COMM_MODE – OLCReject cause waitForCommunicationMode • M3G_E_OLCREJ_INV_DEP_CHAN – OLCReject cause invalidDependentChannel • M3G_E_OLCREJ_REP_FOR_REJ – OLCReject cause replacementForRejected • M3G_E_OLCREJ_SECURITY_DENIED – OLCReject cause securityDenied

This function is only supported in asynchronous mode.

Termination Events

M3GEV_RESPOND_TO_LC_CMPLT
Indicates either the specified OpenLogicalChannelAck response or the OpenLogicalChannelReject response was successfully sent to the remote 3G-324M endpoint.

M3GEV_RESPOND_TO_LC_FAIL
Indicates either the specified OpenLogicalChannelAck response or the OpenLogicalChannelReject response failed to be sent to the remote 3G-324M endpoint. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

- It is invalid to call this function with a board, audio or video device type handle.
- It is invalid to call this function unless an M3GEV_REMOTE_OLC_RCVD event was received on this control device.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the                   */
/*                   applicable control, audio and or video devices          */
/*                   associated with the 3G-324M bearer channel.             */
/*                3) The control, audio, and or video devices have all been  */
/*                   interconnected to their respective network and ipm or   */
/*                   mm devices using the dev_PortConnect( ) or              */
/*                   dev_Connect( )functions.                                */
/*                4) The default simultaneous caps table has been set using  */
/*                   the m3g_GetLocalCap( ) and m3g_SetTCS( ) functions      */
/*                   (not shown).                                            */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
```

```
        */

        /* Received OLC from remote 3G-324M endpoint:  */
        case M3GEV_REMOTE_OLC_RCVD:
        {
           /* Assume application defined its device structure: */
           MYDEV *pMyDev;
           M3G_REMOTE_OLC_REQ *pOLCReq = (M3G_REMOTE_OLC_REQ*) pSRLEvtData;

           m3g_GetUserInfo(devH, &pMyDev);

           /* If the requested capability is satisfactory for reverse direction to receive */
           if(true == isRequestedCapsOK(pMyDev, pOLCReq))
           {
              /* Respond with OLCAck: */
              if (M3G_ERROR == m3g_RespondToOLC(pMyDev->controlDevH,
                                                pOLCReq->logicalChannelNumber,
                                                M3G_E_OLCACK))
      {
         log("Error:  m3g_RespondToOLC(%s)failed - %s\n",
              ATDV_NAMEP(pMyDev->audioDevH), ATDV_ERRMSGP(pMyDev->audioDevH));
         /* handle error… */
      }
      /* Check if audio dataType: */
      if(M3G_E_AUDIO_CAPABILITY == pOLCReq->capabilityType)
      {
         setLCNMediaType(pMyDev, pOLCReq->logicalChannelNumber, AUDIO);
         pMyDev->audioReverseCap = pOLCReq->mediaCapability;
      }
      /* Else if video dataType: */
      else if(M3G_E_VIDEO_CAPABILITY == pOLCReq->capabilityType)
      {
         setLCNMediaType(pMyDev, pOLCReq->logicalChannelNumber, VIDEO);
         pMyDev->videoReverseCap = pOLCReq->mediaCapability;
      }
   }
   else /* Not requested reverse capability not acceptable. */
          {
              /* Respond with OLCReject: */
              if (M3G_ERROR == m3g_RespondToOLC(pMyDev->controlDevH,
                                                pOLCReq->logicalChannelNumber,
                                                M3G_E_OLCREJ_DATATYP_NOT_SUP))
      {
         log("Error:  m3g_RespondToOLC(%s)failed - %s\n",
              ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
         /* handle error… */
      }
   }
   break;
}

        /* Successful m3g_RespondToOLC termination: */
        case M3GEV_RESPOND_TO_LC_CMPLT:
        {
           /* Assume application defined its device structure: */
           MYDEV * pMyDev;
           M3G_LOGICAL_CHANNEL_NUMBER lcn = *(M3G_LOGICAL_CHANNEL_NUMBER *) pSRLEvtData;

           m3g_GetUserInfo(devH, &pMyDev);

           /* Must determine if this was for our audio or video OLCAck (not shown): */
           if (AUDIO == getLCNMediaType(pMyDev, lcn))
           {
               pMyDev->isAudioRevOLCAcked = true;
               pMyDev->revAudioLCN = lcn;
               /* If OLCs have been acknowledged in both forward and reverse directions */
               if (true == isBothAudioOLCsComplete(pMyDev))
```

```
                {
                   /* start media for this device (not shown)… */
                   startAudioMedia(pMyDev));
                }
             }
             else   /* else video: */
             {
                 pMyDev->isAudioRevOLCAcked = true;
                 pMyDev->revAudioLCN = logicalChannelNumber;
                 /* If OLCs have been acknowledged in both forward and reverse directions */
                 if (true == isBothVideoOLCsComplete(pMyDev))
                 {
                    /* start media for this device (not shown)… */
                    startVideoMedia(pMyDev));
                 }
             }
 }

        /* m3g_RespondToOLC Failure indication: */
        case M3GEV_RESPOND_TO_OLC_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_RESPOND_TO_OLC_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
    }
}
```

See Also

- **m3g_OpenLC( )**

m3g_SendH245MiscCmd( )

Name: int m3g_SendH245MiscCmd (deviceHandle, pMiscCmd)

Inputs: SRL_DEVICE_HANDLE deviceHandle • control device handle

M3G_H245_MISC_CMD *pMiscCmd • pointer to M3G_H245_MISC_CMD structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_SendH245MiscCmd( )** function sends the specified H.245 MiscellaneousCommand message to the remote 3G-324M endpoint.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
pMiscCmd	points to the M3G_H245_MISC_CMD structure which specifies the command types. For more information, see the M3G_H245_MISC_CMD structure description.

The **h245MiscCmdType** parameter within the M3G_H245_MISC_CMD structure identifies the type of MiscellaneousCommand specified within the M3G_H245_MISC_CMD_PARAMS union.

When an H.245 MiscellaneousCommand is received from the remote 3G-324M endpoint, an M3GEV_H245_MISC_CMD_RCVD event type is queued to the application. Call the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to cast the returned value to the M3G_H245_MISC_CMD structure.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_SEND_H245_MISC_CMD_CMPLT
Indicates the specified H.245 MiscellaneousCommand indication message was sent successfully.

M3GEV_SEND_H245_MISC_CMD_FAIL
Indicates the specified H.245 MiscellaneousCommand indication message failed to be sent. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

It is invalid to call this function prior to completion of the H.245 Terminal Capability Set (TCS) and H.245 Master Slave Determination (MSD) exchange. This completion is identified by receipt of the M3GEV_REMOTE_TCS_RCVD event for the remote capability acknowledgement, the M3GEV_LOCAL_TCS_ACKD event for the local capability acknowledgement, and the M3GEV_MSD_ESTABLISHED event indicating completion of MSD exchange.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the                   */
/*                   control and audio and media devices.                    */
/*                3) The H.245 forward and reverse logical channels for this */
/*                   video device have been opened successfully and is       */
/*                   streaming to/from the remote H.223 endpoint.            */

/* send H.245 MiscellaneousCommand type videoFastUpdatePicture to remote 3G-324M endpoint: */
int sendFastUpdate(MYDEV pMyDev)
{
   M3G_H245_MISC_CMD    h245MiscCmdBuf = {0};
   h245MiscCmdBuf.version = M3G_LIBRARY_VERSION;
   h245MiscCmdBuf.h245MiscCmdType = M3G_E_FAST_UPDATE_PICTURE;
   /* Note fastUpdatePicture requires no additional parameters in h245MiscCmdParams element */
   if (M3G_ERROR == m3g_SendH245MiscCmd(pMyDev->controlDevH, &h245MiscCmdBuf))
   {
      log("Error:  m3g_SendH245MiscCmd(%s)failed - %s\n",
           ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of sendFastUpdate */

.
.
.
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
```

```
.
void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_SendH245MiscCmd termination: */
        case M3GEV_SEND_H245_MISC_CMD_CMPLT:
            log("M3GEV_H24_UII_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_SendH245MiscCmd Failure indication: */
        case M3GEV_SEND_H245_MISC_CMD_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_SEND_MISC_CMD_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        /* Received H.245 MiscCmd message from a remote 3G-324M endpoint: */
        case M3GEV_H245_MISC_CMD_RCVD:
        {
           /* Assume application defined its device structure: */
           MYDEV * pMyDev;
           M3G_H245_MISC_CMD* pMiscCmd = (M3G_H245_MISC_CMD*) pSRLEvtData;
           m3g_GetUserInfo(devH, &pMyDev);

           log("M3GEV_H245_MISC_CMD_RCVD for device = %s\n",
                ATDV_NAMEP(devH));

           /* Process MiscCmd (not shown): */
           processMiscCmd(pMyDev, pMiscCmd);
           break;

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

None.

m3g_SendH245UII( )

Name: int m3g _SendH245UII (deviceHandle, pH245UII)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control device handle

M3G_H245_UII *pH245UII
- pointer to M3G_H245_UII structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_SendH245UII( )** function sends DTMF (alphanumeric) digits in an H.245 UserInputIndication message to the remote 3G-324M endpoint. The digit buffer format and content are specified in the M3G_H245_UII structure.

Call this function only after H.245 MasterSlaveDetermination transactions and H.245 TerminalCapabilitySet transactions have completed in each direction with the remote 3G-324M endpoint.

When DTMF digits are received from the remote 3G-324M endpoint, an M3GEV_H245_UII_RCVD event is queued to the application. Upon receiving this event, call the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** and cast the returned value to an M3G_H245_UII structure containing the digit string received from the remote.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
pH245UII	points to an M3G_H245_UII structure containing the digit string buffer

This function is only supported in asynchronous mode.

Termination Events

M3GEV_SEND_H245_UII_CMPLT
Indicates the specified alphanumeric digit string was sent successfully.

M3GEV_SEND_H245_UII_FAIL
Indicates the specified alphanumeric digit string failed to be sent. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

It is invalid to call this function prior to completion of MasterSlaveDetermination (MSD) transactions and TerminalCapabilitySet (TCS) transactions exchange. This TerminalCapabilitySet completion is identified by receipt of both the M3GEV_REMOTE_TCS_RCVD event for the remote capabilities and the M3GEV_LOCAL_TCS_ACKD event for the acknowledgement of the local capabilities.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the                   */
/*                   control and audio and media devices.                    */
/*                3) The H.245 forward logical channel for this audio device */
/*                   has been opened successfully and the audio is streaming */
/*                   to/from the H.223 multiplex.                            */

/* User has input DTMF, so send H.245 UII message containing digits to remote 3G-324M endpoint:
*/
int sendUIIDigits(MYDEV pMyDev, const char* digitBuf)
{
   M3G_H245_UII    h245UIIBuf;
   int numDigits = strlen(digitBuf);
   h245UIIBuf.version = M3G_LIBRARY_VERSION;
   h245UIIBuf.numDIgits = numDigits < MAX_NUM_DIGITS ? numDigits : MAX_NUM_DIGITS;
   memcpy(h245UIIBuf.digitBuffer, digitBuf, h245UIIBuf.numDIgits);
   if (M3G_ERROR == m3g_SendH245UII(pMyDev->controlDevH, &h245UIIBuf))
   {
      log("Error:  m3g_SendH245UII (%s)failed - %s\n",
           ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of sendUIIDigits */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
```

```
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_SendH245UII termination: */
        case M3GEV_SEND_H245_UII_CMPLT:
            log("M3GEV_H24_UII_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_SendH245UII Failure indication: */
        case M3GEV_SEND_H245_UII_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_SEND_H245_UII_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        /* Received UII digits from remote 3G-324M endpoint: */
        case M3GEV_H245_UII_RCVD:
        {
           M3G_H245_UII* pH245UIIBuf = (M3G_H245_UII*) pSRLEvtData;
           /* Assume application defined its device structure: */
           MYDEV * pMyDev;
           m3g_GetUserInfo(devH, &pMyDev);

           log("M3GEV_H245_UII_RCVD for device = %s\n",
                ATDV_NAMEP(devH));

           /* Process digit(s) prompt (not shown): */
           processDigitPrompt(pMyDev, pH245UIIBuf);
           break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

None.

m3g_SetParm( )

Name: int m3g_SetParm (deviceHandle, pParmInfo)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- board or control device handle

M3G_PARM_INFO *pParmInfo
- pointer to an M3G_PARM_INFO structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: System Control

Mode: asynchronous

Description

The **m3g_SetParm( )** function sets values for the specified parameter via the M3G_PARM_INFO structure referenced by the **pParmInfo** parameter.

Parameters may be specified for a board device, a control device, or both types of devices. Setting one or more parameters on a board device sets the default values for all control devices associated with that board.

Parameter	Description
deviceHandle	specifies an SRL handle to a board or control device
pParmInfo	points to the M3G_PARM_INFO structure specifying the parameter and its respective value to set. For more information, see Table 3, "M3G_PARM_INFO Parameter Types and Parameter Values", on page 217.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_SET_PARM_CMPLT
Indicates specified parameter values were successfully set. This event does not return any data.

M3GEV_SET_PARM_FAIL
Indicates that the specified events were unsuccessfully set. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

You are responsible for allocating and de-allocating the M3G_PARM_INFO structure referenced by the **pParmInfo** parameter.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.              */
/*                2) m3g_Open( ) has completed opening the board device handle.     */


int setDefaultH245TerminalType(int boardDevH)
{
   /* Set H.245 terminalType to 255 to improve probability of becoming Master */
   M3G_PARM_INFO parmInfo = {0};
   parmInfo.version = M3G_LIBRARY_VERSION;
   parmInfo.parameterType = M3G_E_PRM_H245_TERMINAL_TYPE;
   parmInfo.parmValue. h245TerminalType = 255;

   if (M3G_ERROR == m3g_SetParm(boardDevH, &parmInfo))
   {
      log("Error:  m3g_SetParm(%s)failed - %s\n",
           ATDV_NAMEP(boardDevH), ATDV_ERRMSGP(boardDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of setDefaultH245TerminalType */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
```

```
int devH          = sr_getevtdev();
void *pSRLEvtData = sr_getevtdatap();

switch(evType)
{
    /*
     .
     . Other events not shown…
     .
     */

     /* Successful m3g_SetParm termination: */
     case M3GEV_SET_PARM_EVENTS_CMPLT:
         log("M3GEV_SET_PARM_CMPLT for device = %s\n",
               ATDV_NAMEP(devH));
         break;

     /* m3g_SetParm Failure indication: */
     case M3GEV_SET_PARM_FAIL:
     {
         M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
         log("ERROR:  M3GEV_SET_PARM_FAIL for device = %s\n",
               ATDV_NAMEP(devH));
         log("        Error value = %d\n",(int)*pError);

         /* handle error…*/
         break;
     }

     default:
        printf("Received unknown event = %d for device = %s\n",
                evType, ATDV_NAMEP(devH));

        break;
}
}
```

See Also

- **m3g_GetParm( )**

m3g_SetTCS( )

Name: int m3g_SetTCS (deviceHandle, numSimultaneousSets, pSimultaneousCapList)

Inputs:

SRL_DEVICE_HANDLE deviceHandle	• control device handle
unsigned short numSimultaneousSets	• number of elements specified in **pSimultaneousCapList**
M3G_SIMULTANEOUS_CAP_SET *pSimultaneousCapList	• pointer to array of M3G_SIMULTANEOUS_CAP_SET structure elements

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_SetTCS( )** function sets the default local set of terminal capabilities in the H.245 TerminalCapabilitySet table using the M3G_SIMULTANEOUS_CAP_SET array. This array contains elements of the default capability settings obtained from multiple calls to the **m3g_GetLocalCaps( )** function, or modified capability settings.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
numSimultaneousSets	specifies the number of elements of type M3G_SIMULTANEOUS_CAP_SET specified in **pSimultaneousCapList** array. Must be set to 1 for this release. Only one audio and one video stream is currently supported for each 3G-324M endpoint.
pSimultaneousCapList	points to an array of M3G_SIMULTANEOUS_CAP_SET elements. Each M3G_SIMULTANEOUS_CAP_SET structure must contain one M3G_CAPS_LIST array of data type M3G_ H223_CAPABILITY, and optionally either one array of type M3G_AUDIO_CAPABILITY, and or one array of type M3G_VIDEO_CAPABILITY.

The H.223, audio, and video capability list elements within the M3G_SIMULTANEOUS_CAP_SET structure are initialized by calling **m3g_GetLocalCaps( )** for each device type.

After the default local capability is set, call **m3g_StartH245( )**, and the 3G-324M protocol stack will first synchronize the H.223 layer and then participate in MasterSlaveDetermination and TerminalCapabilitySet exchanges with the remote 3G-324M endpoint.

A TerminalCapabilitySetAck (or TerminalCapabilitySetReject) response is automatically and implicitly sent to acknowledge an incoming TerminalCapabilitySet from the remote endpoint. When remote terminal capabilities are received in a TerminalCapabilitySet message from the remote endpoint, an M3GEV_REMOTE_TCS_RCVD event is queued to the application. When local terminal capabilities have been positively acknowledged via the remote, an M3GEV_LOCAL_TCS_ACKD event is queued to the application.

While the H.245 specification permits TerminalCapabilitySet messages to be exchanged at any time, no attempt at opening logical channels may be initiated until MasterSlaveDetermination and TerminalCapabilitySet transactions, meaning both a request and a response, have completed in each direction with the remote 3G-324M endpoint. At that point, the intersection of the local and remote terminal capability sets are subsequently retrieved by calling the **m3g_GetMatchedCaps( )** function for the control device and each audio and/or video device handle to retrieve compatible H.223, audio and video capabilities respectively.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_SET_TCS_CMPLT
: Indicates the specified default capability set table was successfully set on this control device.

M3GEV_SET_TCS_FAIL
: Indicates the specified default capability set table failed to be set on this control device. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

- It is invalid to call this function with an audio or video device type handle.
- The **numSimultaneousSets** parameter must be set to 1 for this release.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
```

```
/* Preconditions: 1) 3G-324M Library Session has already been started.         */
/*                2) m3g_Open( ) has completed for the applicable              */
/*                   control, audio and video device type                      */
/*                3) assumes globally defined devTbl[] exists for all devices  */
/*                                                                             */
int getDefaultCaps(int devIndex)
{

   M3G_CAPS_LIST * pLocalCaps = &h223Caps;
   /* Retrieve the default H.233 capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].controlDevH,
                                     NULL));
   {
      log("Error:  m3g_GetLocalCaps(%s) for H.223 failed - %s\n",
           ATDV_NAMEP(devTbl[devIndex].controlDevH),
           ATDV_ERRMSGP(devTbl[devIndex].controlDevH));
      /* handle error… */
   }

   /* Retrieve the default audio capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].audioDevH,
                                     NULL));
   {
      log("Error:  m3g_GetLocalCaps(%s) for audio failed - %s\n",
           ATDV_NAMEP(devTbl[devIndex].audioDevH),
           ATDV_ERRMSGP(devTbl[devIndex].audioDevH));
      /* handle error… */
   }

   /* Retrieve the default video capabilities. */
   if (M3G_ERROR == m3g_GetLocalCaps(devTbl[devIndex].videoDevH,
                                     NULL));
   {
      log("Error:  m3g_GetLocalCaps(%s) for video failed - %s\n",
           ATDV_NAMEP(devTbl[devIndex].videoDevH),
           ATDV_ERRMSGP(devTbl[devIndex].videoDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of getDefaultCaps */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
```

```
.
*/

/* Successful m3g_GetLocalCaps termination: */
case M3GEV_GET_LOCAL_CAPS_CMPLT:
{
    /* Assume application defined its device structure: */
    MYDEV * pMyDev;
    M3G_CAPS_LIST * pLocalCaps = pSRLEvtData;

    log("M3GEV_GET_LOCAL_CAPS_CMPLT for device = %s\n",
          ATDV_NAMEP(devH));

    /* Cache appropriate device type (h223, audio, or video) caps in  */
    /* Simultaneous Caps struct  to send in TCS: */
    m3g_GetUserInfo(devH, &pMyDev);

    switch (pMyDev->myType)
    {
        case H223TYPE:
           pMyDev->bearerChannel.simultaneousCaps.pH223Capabilities = pLocalCaps;
           break;
        case AUDIOTYPE:
           pMyDev->bearerChannel.simultaneousCaps.pAudioCapabilities = pLocalCaps;
           break;
        case VIDEOTYPE:
           pMyDev->bearerChannel.simultaneousCaps.pVideoCapabilities = pLocalCaps;
           break;
    }

    /* If received all local capabilities associated with bearer channel: */
    if ((NULL != pMyDev->bearerChannel.simultaneousCaps.pH223Capabilities) &&
        (NULL != pMyDev->bearerChannel.simultaneousCaps.pAudioCapabilities) &&
         NULL != pMyDev->bearerChannel.simultaneousCaps.pVideoCapabilities))
    {
       /* Set default TCS using optionally customized defaults: */
       /* Note only one audio & video device per bearer channel is supported currently:
*/
       if (M3G_ERROR == m3g_SetTCS(controlDevH, 1,
           &pMyDev->bearerChannel.simultaneousCaps));
       {
          log("Error:  m3g_SetTCS(%s) failed - %s\n",
               ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
          /* handle error… */
       }
    }
    break;
}

/* m3g_ GetLocalCaps Failure indication: */
case M3GEV_GET_LOCAL_CAPS_FAIL:
{
    M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
    log("ERROR:  M3GEV_GET_LOCAL_CAPS_FAIL for device = %s\n",
          ATDV_NAMEP(devH));
    log("        Error value = %d\n",(int)*pError);

    /* handle error…*/
    break;
}

case M3GEV_SET_TCS_CMPLT:
    log("M3GEV_SET_TCS_CMPLT for device = %s\n",
          ATDV_NAMEP(devH));
    /* Assuming event handlers are enabled for relevant MSD and TCS unsolicited events,
*/
    /* We can now initiate the H.245 session: */
    if (M3G_ERROR == m3g_StartH245(devH))
```

```
            {
               log("Error:  m3g_StartH245 (%s) failed - %s\n",
                   ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
               /* handle error… */
            }
            break;


        /* m3g_SetTCS Failure indication: */
        case M3GEV_SET_TCS_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_SET_TCS_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }


       /*
        .
        . Other events not shown…
        .
        */

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_GetLocalCaps( )**
- **m3g_StartH245( )**

m3g_SetVendorId( )

Name: int m3g_SetVendorId (deviceHandle, * pVendorIdInfo)

Inputs: SRL_DEVICE_HANDLE deviceHandle • board device handle

M3G_VENDORID_INFO *pVendorIdInfo • pointer to M3G_VENDORID_INFO structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

The **m3g_SetVendorId( )** function configures the information elements to be encoded in the H.245 VendorIdentification indication message. The H.245 VendorIdentification message is automatically sent during H.245 signaling within the 3G-324M session immediately after the H.245 MasterSlaveDetermination message. This function may only be called on a board device and is used to configure the vendor and product information elements for all 3G-324M interfaces.

Changes to the vendor identification information made with **m3g_SetVendorId( )** are effective until the function is called again or until Dialogic® Services are restarted.

The default values for H.245 VendorIdentification message information elements are as follows:

vendor
3.111.112

productNumber
Dialogic HMP

versionNumber
the current product kernel version number

Note that the remote vendor and product information can be retrieved from the H.245 VendorIdentification message from the remote peer by enabling the M3GEV_REMOTE_VENDORID_EVT_TYP bitmask using **m3g_EnableEvents( )** and by parsing the embedded M3G_VENDORID_INFO structure from any events received of that type.

Parameter	Description
deviceHandle	specifies an SRL handle to a board device
pVendorIdInfo	pointer to the M3G_VENDORID_INFO structure

This function is only supported in asynchronous mode.

Termination Events

M3GEV_SET_VENDORID_CMPLT
: Indicates the specified vendor and product information elements have been successfully configured for the board.

M3GEV_SET_VENDORID_FAIL
: Indicates the specified vendor and product information elements have failed to be configured for the board.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the board device.     */

int setVendorId(int boardDevH)
{
   M3G_VENDORID_INFO vendorIdInfo;
   vendorIdInfo.version = M3G_LIBRARY_VERSION2;
   vendorIdInfo.vendor.oidType = M3G_E_OBJECTID_TYPE;
   vendorIdInfo.vendor.oid.length = 7;
   vendorIdInfo.vendor.oid.objectId[0] = 1;
   vendorIdInfo.vendor.oid.objectId[1] = 2;
   vendorIdInfo.vendor.oid.objectId[2] = 3;
   vendorIdInfo.vendor.oid.objectId[3] = 4;
   vendorIdInfo.vendor.oid.objectId[4] = 5;
   vendorIdInfo.vendor.oid.objectId[5] = 6;
   vendorIdInfo.vendor.oid.objectId[6] = 7;
   vendorIdInfo.productNumber.length = 6;
   vendorIdInfo.productNumber.octet[0] = 5;
   vendorIdInfo.productNumber.octet[1] = 4;
   vendorIdInfo.productNumber.octet[2] = 3;
   vendorIdInfo.productNumber.octet[3] = 2;
   vendorIdInfo.productNumber.octet[4] = 1;
   vendorIdInfo.versionNumber.octet[5] = 0;
   vendorIdInfo.productNumber.length = 4;
   vendorIdInfo.versionNumber.octet[0] = 6;
   vendorIdInfo.versionNumber.octet[1] = 0;
   vendorIdInfo.versionNumber.octet[2] = 2;
   vendorIdInfo.versionNumber.octet[3] = 0;
```

```
   if (M3G_ERROR == m3g_SetVendorId(boardDevH, &vendorIdInfo))
   {
      log("Error:  m3g_SetVendorId(%s)failed - %s\n",
           ATDV_NAMEP(boardDevH), ATDV_ERRMSGP(boardDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of setVendorId */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_SetVendorId termination: */
        case M3GEV_SET_VENDORID_CMPLT:
            log("M3GEV_SET_VENDORID_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_SetVendorId Failure indication: */
        case M3GEV_SET_VENDORID_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_START_TRACE_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

None.

m3g_Start( )

Name: int m3g_Start (M3G_START_STRUCT* pStartParams)

Inputs: pStartParams
- pointer to the 3G-324M library startup structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: synchronous

Description

The **m3g_Start( )** function starts and initializes the 3G-324M library. Session configuration information is specified in the M3G_START_STRUCT data structure pointed to by this function.

Parameter	Description
pStartParams	points to the M3G_START_STRUCT structure which specifies 3G-324M session configuration settings

This function is only supported in synchronous mode. The function returns M3G_SUCCESS if the 3G-324M library was successfully started; otherwise, it returns M3G_ERROR on failure.

Cautions

This function must be called and successfully complete, before any other 3G-324M library function is called.

Errors

If this function fails with M3G_ERROR, it is recommended that you verify the following:

- Dialogic services are successfully started with licensed 3G-324M devices in the system.
- The number of devices specified in M3G_START_STRUCT, numEndpoints field, does not exceed the number of licensed 3G-324M endpoints in the system installation.

Example

```
/* Header Files */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
```

```
/* Start 3G-324M Library Session.                                          */
/* Preconditions: 3G-324M Library Session has not yet been started.        */

int start3G324M()
{
   M3G_START_STRUCT myStartParams;
   INIT_M3G_START_STRUCT(M3G_START_STRUCT &myStartParams);
   /* Allocate one E1 trunk for 3G-324M endpoints. */
   myStartParams.numEndpoints = 30;

   /* Start 3G-324M library session. */
   if (M3G_ERROR == m3g_Start(&myStartParams))
   {
      /* handle error… */
   }

   return SUCCESS;
} /* End of start3G324M */
```

See Also

- **m3g_Stop( )**

m3g_StartH245( )

Name: int m3g_StartH245 (deviceHandle, pH223Params)

Inputs: SRL_DEVICE_HANDLE deviceHandle — • control device handle

M3G_H223_SESSION pH223Params — • pointer to M3G_H223_SESSION structure

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

Before any 3G-324M signaling may take place, the respective call must be established and connected using signaling means outside the 3G-324M protocol and this API library.

Once the call is established, the **m3g_StartH245( )** function initiates the H.223 multiplex and demultiplex using the specified parameters. Next, it initiates H.245 message transaction sequence, starting with the MasterSlaveDetermination and TerminalCapabilitySet transaction exchanges.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device
pH223Params	points to an M3G_H223_SESSION structure specifying H.223 multiplex layer parameters. To use MONA procedures, enable MONA support in the M3G_H223_SESSION structure.

Only call this function on a control device after it has been opened and successfully connected using **dev_Connect( )** and **dev_PortConnect( )** functions, in the device management API library.

After successful completion of this function, the application must receive the M3GEV_FRAMING_ESTABLISHED event indicating the framing layer is sufficient to establish the H.223 Abstraction Layer and its upper service access points. Until then, most subsequent H.223 or H.245 related function calls on this control device will fail with an error code of M3G_E_ERR_INV_STATE.

Conversely, at any point after this function completes, the application could receive the M3GEV_FRAMING_LOST event indicating an error condition has occurred in the framing layer which prevents proper functioning of the H.223 multiplex layer. No further 3G-324M functions may be called or complete successfully until the error condition is resolved as signified by the

M3GEV_FRAMING_ESTABLISHED event. Until then, most subsequent H.223 or H.245 related function calls on this control device will fail with an error code of M3G_E_ERR_INV_STATE.

After successful completion of this function, the application should also be prepared to receive the M3GEV_MSD_ESTABLISHED event indicating the result of the MasterSlaveDetermination transactions which transpire only after calling this function.

Any time an H.245 session is active between the local and remote 3G-324M endpoint, the application must also be prepared to receive the unsolicited M3GEV_ENDSESSION_RCVD event. This event indicates an H.245 EndSession command has been received from the remote 3G-324M endpoint and as a result, the current H.245 session has been terminated.

Should the MasterSlaveDetermination transactions fail for any reason, the application should receive the M3GEV_MSD_FAILED event. Upon receiving an M3GEV_FRAMING_LOST, M3GEV_MSD_FAILED, or M3GEV_ENDSESSION_RCVD event, the next action from the application should be to close H.245 session and re-start H.245 sequence via **m3g_StopH245( )** and **m3g_StartH245( )**, respectively.

Similarly, both the M3GEV_REMOTE_TCS_RCVD and the M3GEV_LOCAL_TCS_ACKD events indicate successful exchange of TerminalCapabilitySet transactions for each endpoint.

The application is responsible for allocating and de-allocating the M3G_H223_SESSION buffer referenced by the **pH223Params** pointer parameter.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_START_H245_CMPLT
: Indicates the H.245 protocol has been successfully initiated. This event does not imply any status regarding the H.223 framing layer, H.245 MasterSlaveDetermination, nor H.245 TerminalCapabilitySet exchange.

M3GEV_START_H245_FAIL
: Indicates the H.245 protocol has failed to initiate. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the applicable        */
/*                   control, audio and or video devices                     */
/*                   associated with the 3G-324M bearer channel.             */
/*                3) The control device has been interconnected to its       */
/*                   respective network timeslot or Nb UP device interface   */
/*                   using the dev_Connect( ) or dev_PortConnect( )          */
/*                   functions respectively.                                 */
/*                4) The audio or video devices have been connected to their */
/*                   respective ipm or mm devices which source or sink the   */
/*                   the media stream using the dev_PortConnect( ) function. */
/*                5) The default simultaneous caps table has been set using  */
/*                   the m3g_GetLocalCaps( ) and m3g_SetTCS( )               */
/*                   function (not shown)                                    */
/*                6) Event handlers have been enabled for the following      */
/*                   events (not shown):                                     */
/*                   M3GEV_START_H245_CMPLT,                                 */
/*                   M3GEV_START_H245_FAIL,                                  */
/*                   M3GEV_FRAMING_ESTABLISHED,                              */
/*                   M3GEV_FRAMING_LOST,                                     */
/*                   M3GEV_MSD_ESTABLISHED,                                  */
/*                   M3GEV_MSD_FAILED,                                       */
/*                   M3GEV_REMOTE_TCS_RCVD                                   */
/*                   M3GEV_LOCAL_TCS_ACKD                                    */
/*                   M3GEV_ENDSESSION_RCVD                                   */
/*                   M3GEV_ENDSESSION_SENT                                   */
/*                   M3GEV_SEND_MONA_PREF_MSG                                */
/*                   M3GEV_MONA_PREF_MSG _RCVD                               */
/*                   M3GEV_TX_MPC_ESTABLISHED                                */
/*                   M3GEV_RX_MPC_ESTABLISHED                                */


int startH245(int controlDevH)
{
   M3G_H223_SESSION h223Params =
   {
      M3G_LIBRARY_VERSION,    /* version              */
      M3G_E_H223_MUX_LEVEL2,  /* defaultH223MuxLevel  */
      254,                    /* maxALSDUSize         */
      M3G_TRUE,               /* isWNSRPEnabled       */
      M3G_TRUE,               /* isMultipleMsgsPerPdu */
      M3G_TRUE                /* isMONAEnabled        */
   }

   /* Initiate the H.245 session. */
   if (M3G_ERROR == m3g_StartH245(controlDevH, &h223Params))
   {
      log("Error:  m3g_StartH245(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of startH245 */
.
.
.
```

```
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_StartH245 termination: */
        case M3GEV_START_H245_CMPLT:
            log("M3GEV_START_H245_CMPLT for device = %s\n",
                ATDV_NAMEP(devH));
            /* Device must receive M3GEV_FRAMING_ESTABLISHED before it  */
            /* can participate in MasterSlaveDetermination exchange.    */
            break;

        /* m3g_StartH245 Failure indication: */
        case M3GEV_START_H245_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_START_H245_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        /* Framing layer is successfully established between local and       */
        /* remote 3G-324M endpoint.                                          */
        case M3GEV_FRAMING_ESTABLISHED:
            log("M3GEV_FRAMING_ESTABLISHED for device = %s\n",
                ATDV_NAMEP(devH));
            /* Device is now ready to participate in exchange of             */
            /* MasterSlaveDetermination message and TerminalCapabilitySet    */
            /* messages once local terminal capabilities are specified.      */
            break;

        /* Framing layer failed to establish between local and remote        */
        /* 3G-324M endpoint.                                                 */
        case M3GEV_FRAMING_LOST:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_FRAMING_LOST for device = %s\n",
                ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
```

```
  }

  /* MasterSlaveDetermination transactions completed between local and */
  /* remote 3G-324M endpoints.                                         */
  case M3GEV_MSD_ESTABLISHED:
  {
     const char* msdStr[] =
     {
        "M3G_E_H245_MASTER",
        "M3G_E_H245_SLAVE",
        "M3G_E_H245_IDENTICAL_MSD_NUMBERS"
     };
     M3G_E_H245_MSD_RESULT msdResult = *(M3G_E_H245_MSD_RESULT*)pSRLEvtData;
     log("Device %s MSD result: =%s\n", ATDV_NAMEP(devH), msdStr[msdResult]);
     break;
   }

  /* Error in MasterSlaveDetermination between local and remote        */
  /* 3G-324M endpoint.                                                 */
  case M3GEV_MSD_FAILED:
  {
      M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
      log("ERROR:  M3GEV_MSD_FAILED for device = %s\n",
                   ATDV_NAMEP(devH));
      log("        Error value = %d\n",(int)*pError);
      /* handle error…*/
      break;
  }

  /* Received TCS from remote 3G-324M endpoint:  */
  case M3GEV_REMOTE_TCS_RCVD:
  {
     /* Assume application defined its device structure: */
     MYDEV * pMyDev;
     log("M3GEV_REMOTE_TCS_RCVD for device = %s\n",
          ATDV_NAMEP(devH));
     /* If both local and remote TCS transactions have completed, can */
     /* initiate the opening of logical channels.                     */
     /* Cache this TCS transaction completion */
     m3g_GetUserInfo(devH, &pMyDev);
     pMyDev->isRemoteTCSCompleted = true;

     /* If both remote and local TCS transactions complete: */
     if(pMyDev->isLocalTCSCompleted)
     {
        /* Open any transmit media channels that may not have */
        /* been established via MONA MPC procedures */
       if(pMyDev->audioMPCEstablished == false)
       {
         openAudioLogicalChannel(pMyDev);
       }
       if(pMyDev->videoMPCEstablished == false)
       {
         openVideoLogicalChannel(pMyDev);
       }
   }
   break;
}

  /* Received TCSAck from remote 3G-324M endpoint: */
  case M3GEV_LOCAL_TCS_ACKD:

 {
  /* Assume application defined its device structure: */
  MYDEV * pMyDev;
  log("M3GEV_REMOTE_TCS_ACKD for device = %s\n",
  ATDV_NAMEP(devH));
```

```
   /* If both local and remote TCS transactions have completed, can */
   /* initiate the opening of logical channels. */
   /* Cache this TCS transaction completion */

   m3g_GetUserInfo(devH, &pMyDev);
   pMyDev->isLocalTCSCompleted = true;

   /* If both remote and local TCS transactions complete: */
   if(pMyDev->isRemoteTCSCompleted)

   {
      /* Open any transmit media channels that may not have */
      /* been established via MONA MPC procedures           */
      if(pMyDev->audioMPCEstablished == false)
      {
         openAudioLogicalChannel(pMyDev);
      }
      if(pMyDev->videoMPCEstablished == false)
      {
         openVideoLogicalChannel(pMyDev);
      }
   }
   break;
}
case M3GEV_SEND_MONA_PREF_MSG:
case M3GEV_MONA_PREF_MSG_RCVD:
   {
      M3G_MONA_TXRX_MPC_SUPPORT* pMPC = (M3G_MONA_TXRX_MPC_SUPPORT*)pSRLEvtData;
      log("MONA Pref_Msg %s, rxMPC:0x%x txMPC:0x%x\n",
         (evType == M3GEV_SEND_MONA_PREF_MSG) ? "sent" : "rcvd",
         pMPC->rxMPCMask, pMPC->txMPCMask);
      break;
   }
case M3GEV_TX_MPC_ESTABLISHED:
case M3GEV_RX_MPC_ESTABLISHED:
   {
      M3G_MONA_MPC* pMPC = (M3G_MONA_MPC*)pSRLEvtData;
      M3G_LOGICAL_CHANNEL_NUMBER lcn = pMPC->logicalChannelNumber;
      M3G_E_DIRECTION direction=(M3GEV_TX_MPC_ESTABLISHED==evtType)? M3G_E_TX : M3G_E_RX;
      M3G_E_CAPABILITY mediaType = pMPC->capabilityType;

      /* Assume application defined its device structure: */
      MYDEV * pMyDev;
      m3g_GetUserInfo(devH, &pMyDev);
      log("MPC established: %s %s LCN:%d\n",
          (M3G_E_TX == direction) ? "TX" : "RX",
          (M3G_E_AUDIO_CAPABILITY == mediaType) ? "AUDIO" : "VIDEO",
          lcn);
      if (M3G_E_AUDIO_CAPABILITY == pMPC->capabilityType)

      {
         /* Cache that this TX audio MPC is established so it need not be opened */
         /* via legacy H.245 openLogicalChannel procedures via m3g_OpenLC( ) */

         if (M3G_E_TX == direction)
         {
            pMyDev->audioMPCEstablished = true;
         }

         /* Activate audio streaming in specified direction */
         ActivateAudioMedia(direction);
      }
      else
      {
```

```
            /* Cache that this TX video MPC is established so it need not be opened */
            /* via legacy H.245 openLogicalChannel procedures via m3g_OpenLC( )     */
            if (M3G_E_TX == direction)
            {
               pMyDev->videoMPCEstablished = true;
            }

            /* Activate video streaming in specified direction */
            ActivateVideoMedia(direction);
         }
         break;
      }
   default:
      printf("Received unknown event = %d for device = %s\n",
             evType, ATDV_NAMEP(devH));
      break;
   }
}
```

See Also

- **m3g_StopH245( )**

m3g_StartMedia( )

Name: int m3g_StartMedia (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- audio or video device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: Data Flow

Mode: asynchronous

Description

The **m3g_StartMedia( )** function starts transmitting and/or receiving media streams between the specified media device and the H.223 multiplex/demultiplex.

Parameter	Description
deviceHandle	specifies an SRL handle to an audio or video device

Only call this function after the required H.245 forward/reverse logical channels or MONA media preconfigured channels (MPC) have been successfully opened between the local and the remote 3G-324M endpoints.

The direction of media initiated is determined by the logical channel opened at the time of the function call. If the forward logical channel for this device is successfully opened, the transmit direction is enabled. If the reverse logical channel is successfully opened, the receive direction is enabled. If both forward and reverse logical channels are successfully opened, bi-directional media flow is enabled.

The actual R4 device type that transmits or receives the associated audio or video data streams, such as an ipmBxCy or an mmBxCy device, must be interconnected on the packet stream via **dev_PortConnect( )** prior to calling this function. If connecting to a PCM network device such as an dtiBxTy or dxxxBxCy device, this must be connected via the **dev_Connect( )** function prior to calling this function.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_START_MEDIA_CMPLT
Indicates streaming between the H.223 aggregate and the specified media device has been successfully initiated.

M3GEV_ START_MEDIA_FAIL

Indicates streaming between the H.223 aggregate and the specified media device has failed to initiate. The error code is included in the event as detailed in Chapter 13, “Events”.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the                   */
/*                   applicable control, audio and or video devices          */
/*                   associated with the 3G-324M bearer channel.             */
/*                3) The control, audio, and or video devices have all been  */
/*                   interconnected to their respective network and ipm or   */
/*                   mm devices using the dev_PortConnect( ) or              */
/*                   dev_Connect( )functions.                                */
/*                4) H.245 MasterSlaveDetermination and                      */
/*                   TerminalCapabilitySet transaction exchange has          */
/*                   completed (not shown).                                   */
.
.
.
/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
```

```
. Other events not shown…
.
*/

/* Received TCS from remote 3G-324M endpoint:  */
case M3GEV_REMOTE_TCS_RCVD:
{
   /* Assume application defined its device structure: */
   MYDEV * pMyDev;

   /* If both local and remote TCS transactions have completed, can */
   /* initiate the opening of logical channels.                     */

   /* Cache this TCS transaction completion */
   m3g_GetUserInfo(devH, &pMyDev);
   pMyDev->isRemoteTCSCompleted = true;

   /* If both remote and local TCS transactions complete: */
   if(true == pMyDev->isLocalTCSCompleted)
   {
      /* Start opening appropriate logical channels */
       startOpeningLogicalChannels(pMyDev);
   }
   break;
}

/* Received TCSAck from remote 3G-324M endpoint:  */
case M3GEV_LOCAL_TCS_ACKD:
{
   /* Assume application defined its device structure: */
   MYDEV * pMyDev;

   /* If both local and remote TCS transactions have completed, can */
   /* initiate the opening of logical channels.                     */

   /* Cache this TCS transaction completion */
   m3g_GetUserInfo(devH, &pMyDev);
   pMyDev->isLocalTCSCompleted = true;

   /* If both remote and local TCS transactions complete: */
   if(true == pMyDev->isRemoteTCSCompleted)
   {
      /* Start opening appropriate logical channels */
       startOpeningLoigicalChannels(pMyDev);
   }
   break;
}

/* Received OLCAck from remote 3G-324M endpoint:  */
case M3GEV_OPEN_LC_CMPLT:
{
   /* Assume application defined its device structure: */
   MYDEV * pMyDev;
   M3G_REMOTE_OLCACK_RESP* pOLCAckResp =
                               (M3G_REMOTE_OLCACK_RESP *) pSRLEvtData;

   /* Cache this TCS transaction completion */
   m3g_GetUserInfo(devH, &pMyDev);

   /* Must determine if this was for our audio or video OLC (not shown): */
   if (true == isCapTypeAudio(&pOLCAckResp->mediaCapability))
   {
      pMyDev->isAudioFwdOLCAcked = true;
      pMyDev->fwdAudioLCN = pOLCAckResp->logicalChannelNumber;
      /* If OLCs have been acknowledged in both forward and reverse directions */
      if (true == isBothAudioOLCsComplete(pMyDev))
      {
```

```
                /* start media for this device (not shown)… */
                startAudioMedia(pMyDev));
             }
          }
          else   /* else video: */
          {
             pMyDev->isVideoFwdOLCAcked = true;
             pMyDev->fwdVideoLCN = pOLCAckResp->logicalChannelNumber;
             /* If OLCs have been acknowledged in both forward and reverse directions */
             if (true == isBothVideoOLCsComplete(pMyDev))
             {
                /* start media for this device (not shown)… */
                startVideoMedia(pMyDev));
             }
          }
          break;
       }

       /* Received OLC from remote 3G-324M endpoint:  */
       case M3GEV_REMOTE_OLC_RCVD:
       {
          /* Assume application defined its device structure: */
          MYDEV * pMyDev;
          M3G_REMOTE_OLC_REQ * pOLCReq = (M3G_REMOTE_OLC_REQ *) pSRLEvtData;

          m3g_GetUserInfo(devH, &pMyDev);

          /* If the requested capability is satisfactory for reverse direction to receive */
          if(true == isRequestedCapsOK(pMyDev, pOLCReq))
          {
             /* Respond with OLCAck: */
             if (M3G_ERROR == m3g_RespondToOLC(pMyDev->controlDevH,
                                               pOLCReq->logicalChannelNumber,
                                               M3G_E_OLCACK))
       {
          log("Error:  m3g_RespondToOLC(%s)failed - %s\n",
               ATDV_NAMEP(pMyDev->audioDevH), ATDV_ERRMSGP(pMyDev->audioDevH));
          /* handle error… */
       }
       /* Check if audio dataType: */
       if (M3G_E_AUDIO_CAPABILITY == pOLCReq->capabilityType)
       {
                pMyDev->revAudioLCN = pOLCReq->logicalChannelNumber;
          pMyDev->revAudioCap = pOLCReq->mediaCapability;
       }
       /* Else if video dataType: */
       else if (M3G_E_VIDEO_CAPABILITY == pOLCReq->capabilityType)
       {
                pMyDev->revVideoLCN = pOLCReq->logicalChannelNumber;
          pMyDev->revVideoCap = pOLCReq->mediaCapability;
       }
    }
    else /* Not requested reverse capability not acceptable. */
          {
              /* Respond with OLCReject: */
              if (M3G_ERROR == m3g_RespondToOLC(pMyDev->controlDevH,
                                                pOLCReq->logicalChannelNumber,
                                                M3G_E_OLCREJ_DATATYP_NOT_SUP))
       {
          log("Error:  m3g_RespondToOLC(%s)failed - %s\n",
               ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
          /* handle error… */
       }
    }
    break;
}
```

```
        /* Successful m3g_RespondToOLC termination: */
        case M3GEV_RESPOND_TO_LC_CMPLT:
        {
           /* Assume application defined its device structure: */
           MYDEV * pMyDev;
           M3G_LOGICAL_CHANNEL_NUMBER lcn = *(M3G_LOGICAL_CHANNEL_NUMBER *) pSRLEvtData;

           m3g_GetUserInfo(devH, &pMyDev);

           /* Must determine if this was for our audio or video OLCAck (not shown): */
           if (AUDIO == getLCNMediaType(pMyDev, lcn))
           {
               pMyDev->isAudioRevOLCAcked = true;
               /* If OLCs have been acknowledged in both forward and reverse directions */
               if (true == isBothAudioOLCsComplete(pMyDev))
               {
                  /* start media for this device (not shown)… */
                  startAudioMedia(pMyDev));
               }
           }
           else   /* else video: */
           {
               pMyDev->isVideoRevOLCAcked = true;
               /* If OLCs have been acknowledged in both forward and reverse directions */
               if (true == isBothVideoOLCsComplete(pMyDev))
               {
                  /* start media for this device (not shown)… */
                  startVideoMedia(pMyDev));
               }
           }
 }

        /* Successful m3g_StartMedia termination: */
        case M3GEV_START_MEDIA_CMPLT:
            log("M3GEV_START_MEDIA_CMPLT for device = %s\n",
                ATDV_NAMEP(devH));
            break;


        /* m3g_StartMedia Failure indication: */
        case M3GEV_START_MEDIA_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_START_MEDIA_FAIL for device = %s\n",
                ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
.
.
.
int startOpeningLogicalChannels(MYDEV *pMyDev)
{
   M3G_CAPS_LIST commonCaps;
   /* Retrieve the common H.233 capabilities: */
   if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->controlDevH, &commonCaps))
   {
      log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
```

```
            ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
         /* handle error… */
      }
      /* Configure the H.223 multiplex parameters for audio OLC (not shown)… */
      setOLCH223MuxParameters(&pMyDev->h223AudioOLCParams, &commonCaps, AUDIO);

      /* Configure the H.223 multiplex parameters for video OLC (not shown)… */
      setOLCH223MuxParameters(&pMyDev-> h223VideoOLCParams, &commonCaps, VIDEO);

      /* Retrieve the common audio capabilities: */
      if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->audioDevH, &commonCaps))
      {
         log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
            ATDV_NAMEP(pMyDev->audioDevH), ATDV_ERRMSGP(pMyDev->audioDevH));
         /* handle error… */
      }
      /* initiate OLC for Tx audio */
      sendAudioOLC(pMyDev, &commonCaps);

      /* Retrieve the common video capabilities: */
      if (M3G_ERROR == m3g_GetMatchedCaps(pMyDev->videoDevH, &commonCaps))
      {
         log("Error:  m3g_GetMatchedCaps(%s)failed - %s\n",
            ATDV_NAMEP(pMyDev->videoDevH), ATDV_ERRMSGP(pMyDev->videoDevH));
         /* handle error… */
      }
      /* initiate OLC for Tx video (not shown)… */
      sendVideoOLC(pMyDev, &commonCaps);

      return SUCCESS;
   } /* End of startOpeningLogicalChannels */


   int sendAudioOLC(MYDEV *pMyDev, M3G_CAPS_LIST* pCommonAudioCaps)
   {
      int index;

      /* Choose the preferred audio capability from among the common types: */
      for(index = 0;
          ((index < pCommonAudioCaps->numCaps) && (index < MAX_CAPABILITIES_PER_DEVICE));
          index++)
      {
          /* Capaibility selection logic not shown */
   if (true == isAudioPreferred(pCommonAudioCaps->capability[index]))
   {
      if (M3G_ERROR == m3g_OpenLC(pMyDev->controlDevH,
                                      pMyDev->h223AudioOLCParams,
                                      M3G_E_AUDIO_CAPABILITY,
                                      &pCommonAudioCaps->capability[index]))
      {
         log("Error:  m3g_OpenLC(%s)failed - %s\n",
            ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
         /* handle error… */
      }
      break;
   }
      } /* endFor */
      return SUCCESS;
   } /* End of sendAudioOLC */


   int sendVideoOLC(MYDEV *pMyDev, M3G_CAPS_LIST* pCommonVideoCaps)
   {
      int index;

      /* Choose the preferred video capability from among common types: */
      for(index = 0;
```

```
        ((index < pCommonVideoCaps->numCaps) && (index < MAX_CAPABILITIES_PER_DEVICE));
        index++)
    {
        /* Capaibility selection logic not shown */
if (true == isVideoPreferred(pCommonVideoCaps->capability[index]))
{
   if (M3G_ERROR == m3g_OpenLC(pMyDev->controlDevH,
                                    pMyDev->h223VideoOLCParams,
                                    M3G_E_VIDEO_CAPABILITY,
                                    &pCommonVideoCaps->capability[index]))
   {
      log("Error:  m3g_OpenLC(%s)failed - %s\n",
           ATDV_NAMEP(pMyDev->controlDevH), ATDV_ERRMSGP(pMyDev->controlDevH));
      /* handle error… */
   }
   break;
}
   } /* endFor */
   return SUCCESS;
} /* End of sendVideoOLC */


int startAudioMedia(MYDEV *pMyDev)
{
   if (M3G_ERROR == m3g_StartMedia(pMyDev->audioDevH))
   {
log("Error:  m3g_StartMedia(%s)failed - %s\n",
    ATDV_NAMEP(pMyDev->audioDevH), ATDV_ERRMSGP(pMyDev->audioDevH));
/* handle error… */
   }
}

int startVideoMedia(MYDEV *pMyDev)
{
   if (M3G_ERROR == m3g_StartMedia(pMyDev->videoDevH))
   {
log("Error:  m3g_StartMedia(%s)failed - %s\n",
    ATDV_NAMEP(pMyDev->videoDevH), ATDV_ERRMSGP(pMyDev->videoDevH));
/* handle error… */
   }
}
```

See Also

- **m3g_StopMedia( )**

m3g_StartTrace( )

Name: int m3g_StartTrace (deviceHandle, *pTraceInfo)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control or board device handle

M3G_TRACE_INFO *pTraceInfo
- pointer to M3G_TRACE_INFO

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: Utility

Mode: asynchronous

Description

The **m3g_StartTrace( )** function initiates and configures 3G-324M tracing to a user-specified log file. This function may be called for a control device or board device only. If called on a board device, tracing is applied to all 3G-324M devices opened on the board. The default location for log files is /usr/dialogic/data.

A post-processing log file parser utility is provided. The parser utility is called m3g_parser and is used as follows from the command line:

```
m3g_parser <logfile>
```

After executing the parser, a separate log file is created for every channel for each category that has been enabled in the specified log file. For details on the categories or levels of tracing, see the M3G_TRACE_INFO structure.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device or board device
pTraceInfo	pointer to M3G_TRACE_INFO structure which contains configuration information for tracing

This function is only supported in asynchronous mode.

Termination Events

M3GEV_START_TRACE_CMPLT
Indicates the specified tracing level(s) has been successfully initiated for the given device(s).

M3GEV_START_TRACE_FAIL
Indicates the specified tracing level(s) for the given device(s) has failed to initiate. The error code is included in the event.

Cautions

3G-324M tracing can only be enabled for either the board device or individual devices. It cannot be enabled and configured at both the board level and individual device level.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.         */
/*                2) m3g_Open( ) has completed opening the board device handle */

int startH245Tracing(int boardDevH)
{
   const char h245LogFileName[] = "/usr/dialogic/log/h245messages.txt";

   M3G_TRACE_INFO traceInfo;
   traceInfo.version = M3G_LIBRARY_VERSION;
   traceInfo.logfile = h245LogFileName;
   traceInfo.bitmask = M3G_TRACE_H245;

   if (M3G_ERROR == m3g_StartTrace(boardDevH, &traceInfo))
   {
      log("Error:  m3g_StartTrace(%s)failed - %s\n",
           ATDV_NAMEP(boardDevH), ATDV_ERRMSGP(boardDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of startH245Tracing */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.
```

```
void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_StartTrace termination: */
        case M3GEV_START_TRACE_CMPLT:
            log("M3GEV_START_TRACE_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_StartTrace Failure indication: */
        case M3GEV_START_TRACE_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_START_TRACE_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_StopTrace( )**

m3g_Stop( )

Name: int m3g_Stop( )

Inputs: none

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gerrs.h

Category: System Control

Mode: synchronous

Description

The **m3g_Stop( )** function stops the 3G-324M library and releases all allocated resources. This function must be the last 3G-324M function called before exiting the application.

This function is only supported in synchronous mode. The function returns M3G_SUCCESS if the 3G-324M library was successfully stopped; otherwise, it returns M3G_ERROR on failure.

Cautions

- The 3G-324M library must have been previously started using **m3g_Start( )** before calling this function.
- Close all devices opened through **m3g_Open( )** before calling this function.
- This function must be the last 3G-324M function called before exiting the application.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Stop 3G-324M Library Session                                                */
/* Preconditions: A 3G-324M library session has already been started via m3g_Start( ) */
```

```
int stop3G324M()
{
   /* Stop the 3G-324M library session */
   if (M3G_ERROR == m3g_Stop())
   {
      /* handle error… */
   }
   return SUCCESS;
} /* End of stop3G324M */
```

See Also

- **m3g_Start( )**

m3g_StopH245( )

Name: int m3g_StopH245 (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: H.245 Control

Mode: asynchronous

Description

After all H.245 message exchange has completed with the remote 3G-324M endpoint, the **m3g_StopH245( )** function terminates the H.245 session. If the local endpoint is the first to terminate the H.245 session, it first sends an EndSession command message. Call this function when the 3G-324M associated call has been disconnected, or upon graceful application termination.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device

Only call this function after the associated H.245 logical channel usage is complete and media channels have been stopped using **m3g_StopMedia( )**. All the associated audio and video media devices may only be disconnected from their respective H.223 multiplex channels after the H.245 processing has completed and all H.223 multiplexing has terminated. At that point, the audio and video devices may be disconnected using **dev_PortDisconnect( )** or **dev_Disconnect( )**, while the control channel may be disconnected using either **dev_Disconnect( )** or **dev_PortDisconnect( )** depending on whether the transport type is the CT bus or an Nb UP interface, respectively.

Any subsequent H.245 transactions on this control device may only take place after successfully restarting the H.245 session via the **m3g_StartH245( )** function.

Any time an H.245 session is active between the local and remote 3G-324M endpoint, be prepared to receive the unsolicited M3GEV_ENDSESSION_RCVD event. This event indicates an H.245 EndSession command has been received from the remote 3G-324M endpoint and as a result, the current H.245 session has been terminated. You must then call **m3g_StopH245( )** to deallocate H.245 session associated resources.

This function is only supported in asynchronous mode.

Termination Events

M3GEV_STOP_H245_CMPLT
Indicates the H.245 session has been successfully terminated.

M3GEV_ STOP_H245_FAIL
Indicates the H.245 session has failed to terminate gracefully. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Code Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.       */
/*                2) m3g_Open( ) has completed opening the applicable        */
/*                   control, audio and or video devices                     */
/*                   associated with the 3G-324M bearer channel.             */
/*                3) An H.245 session has been initiated between the local   */
/*                   and remote 3G-324M endpoint.                            */
/*
int terminate3G324MCall (int controlDevH)
{
   /* Terminate the H.245 session. */
   if (M3G_ERROR == m3g_StopH245(controlDevH))
   {
      log("Error:  m3g_StopH245(%s)failed - %s\n",
           ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of terminate3G324MCall */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
```

```
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Received EndSession message from remote 3G-324M endpoint. */
        case M3GEV_ENDSESSION_RCVD:
            log("M3GEV_ENDSESSION_RCVD for device = %s\n",
                  ATDV_NAMEP(devH));
            /* Must now call m3g_StopH245( ) to deallocate H.245 session resources. */
            /* devices from their respective source or sinks (not shown)…             */
            if (M3G_ERROR == m3g_StopH245(controlDevH))
            {
               log("Error:  m3g_StopH245(%s)failed - %s\n",
               ATDV_NAMEP(devH), ATDV_ERRMSGP(devH));
               /* handle error… */
             }
            break;

        /* Successful m3g_StopH245 termination: */
        case M3GEV_STOP_H245_CMPLT:
            log("M3GEV_STOP_H245_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            /* May now disconnect H.223 multiplex from CT bus and audio and video     */
            /* devices from their respective source or sinks (not shown)…             */
            break;

        /* m3g_StopH245 Failure indication: */
        case M3GEV_STOP_H245_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_STOP_H245_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
   }
}
```

See Also

- **m3g_StartH245( )**

m3g_StopMedia( )

Name: int m3g_StopMedia (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- audio or video device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: Data Flow

Mode: asynchronous

Description

The **m3g_StopMedia( )** function stops transmitting and/or receiving media data streams between the specified media device and the H.223 multiplex/demultiplex.

Parameter	Description
deviceHandle	specifies SRL handle to an audio or video device

This function is only supported in asynchronous mode.

Termination Events

M3GEV_STOP_MEDIA_CMPLT
Indicates streaming between the H.223 aggregate and the specified media device has been successfully terminated.

M3GEV_ STOP_MEDIA _FAIL
Indicates streaming between the H.223 aggregate and the specified media device has failed to gracefully terminate. The error code is included in the event as detailed in Chapter 13, "Events".

Cautions

It is invalid to call this function unless the audio device or video device is currently streaming as the result of a prior call to **m3g_StartMedia( )** or **m3g_ModifyMedia( )**.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, "Error Codes".

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library H.245 forward and reverse logical channel have      */
/*                   already been established (not shown).                               */
/*                2) Call associated with bearer channel disconnected (not shown).      */
int handleDisconnectedCall(MYDEV * pMyDev)
{

   /* Disconnect and stop the media (not shown): */
   disconnectAndStopMedia(pMyDev);

   /* Close audio forward logical channel */
   if (M3G_ERROR == m3g_CloseLC(pMyDev->controlDevH, pMyDev->audioLCN,
M3G_E_REQ_CHAN_CLOSE_NORMAL))
   {
      log("Error:  m3g_CloseLC(%s)failed - %s\n",
           ATDVNAMEP(pMyDev->controlDevH), ATDV_LASTERR(pMyDev->controlDevH));
      /* handle error… */
   }

   /* Close video forward logical channel */
   if (M3G_ERROR == m3g_CloseLC(pMyDev->controlDevH, pMyDev->videoLCN,
3G_E_REQ_CHAN_CLOSE_NORMAL))
   {
      log("Error:  m3g_CloseLC(%s)failed - %s\n",
           ATDVNAMEP(pMyDev->controlDevH), ATDV_LASTERR(pMyDev->controlDevH));
      /* handle error… */
   }

   /* Stop the H.245 Session (not shown): */
   stopH245(pMydev);

}  /* End of handleDisconnectedCall */

int disconnectAndStopMedia(MYDEV * pMyDev)
{
   /* Disconnect audio devices: */
   dev_PortDisconnect(pMyDev->audioDevH,pMyDev->audioPortList, pMyDev);
   dev_PortDisconnect(pMyDev->ipmAud, pMyDev->ipmPortList, pMyDev);

   /* Stop streaming between audio i/o and H.223 multiplex: */
   if (M3G_ERROR == m3g_StopMedia(pMyDev->audioDevH))
   {
      log("Error:  m3g_StopMedia(%s)failed - %s\n",
           ATDVNAMEP(pMyDev->audioDevH), ATDV_LASTERR(pMyDev->audioDevH));
      /* handle error… */
   }

   /* Disconnect video devices: */
   dev_PortDisconnect(pMyDev->videoDevH,pMyDev->videoPortList, pMyDev);
   dev_PortDisconnect(pMyDev->ipmVid, pMyDev->ipmPortList, pMyDev);

   /* Stop streaming between video i/o and H.223 multiplex: */
   if (M3G_ERROR == m3g_StopMedia(pMyDev->video DevH))
   {
      log("Error:  m3g_StopMedia(%s)failed - %s\n",
```

```
            ATDVNAMEP(pMyDev->audioDevH), ATDV_LASTERR(pMyDev->videoDevH));
         /* handle error… */
      }
   }  /* End of disconnectAndStopMedia */
   .
   .
   .
   /* SRL event handler: */
      for (;;)
      {
         if (-1 != sr_waitevt(500))
               process_event();
      }
   .
   .
   .
   void process_event(void)
   {
      /* process the SRL events */
      int evType        = sr_getevttype();
      int devH          = sr_getevtdev();
      void *pSRLEvtData = sr_getevtdatap();

      switch(evType)
      {
          /*
           .
           . Other events not shown…
           .
           */

           /* Successful m3g_CloseLC termination: */
           case M3GEV_CLOSE_LC_CMPLT:
           {
              /* Assume application defined its device structure: */
              MYDEV * pMyDev;
              M3G_LOGICAL_CHANNEL_NUMBER lcn = *(M3G_LOGICAL_CHANNEL_NUMBER *) pSRLEvtData;
              m3g_GetUserInfo(devH, &pMyDev);

              /* Determine whether lcn is for audio or video: */
              if(AUDIO == getLCNMediaType(pMyDev, lcn))
              {
                  pMyDev->isAudioFwdOLCAcked = false;
                  pMyDev->fwdAudioLCN = 0;
              }
              else   /* else video: */
              {
                  pMyDev->isVideoFwdOLCAcked = false;
                  pMyDev->fwdVideoLCN = 0;
              }

              /* If both audio and video CLCAcks received: */
              if ((!pMyDev->isAudioFwdOLCAcked) && (!pMyDev->isVideoFwdOLCAcked))
              {
                 /* Stop the H.245 Session (not shown): */
                 stopH245(pMydev);
              }
       break;
    }

           /* m3g_CLoseLC Failure indication: */
           case M3GEV_CLOSE_LC_FAIL:
           {
               M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
               log("ERROR:  M3GEV_CLOSE_LC_FAIL for device = %s\n",
                     ATDV_NAMEP(devH));
               log("        Error value = %d\n",(int)*pError);
```

```
            /* handle error…*/
            break;
        }

        /* Successful m3g_StopMedia termination: */
        case M3GEV_STOP_MEDIA_CMPLT:
            log("M3GEV_STOP_MEDIA_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_ StopMedia Failure indication: */
        case M3GEV_STOP_MEDIA_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_STOP_MEDIA_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);

            /* handle error…*/
            break;
        }

        default:
           printf("Received unknown event = %d for device = %s\n",
                   evType, ATDV_NAMEP(devH));
           break;
    }
}
```

See Also

- **m3g_StartMedia( )**

m3g_StopTrace( )

Name: int m3g_StopTrace (deviceHandle)

Inputs: SRL_DEVICE_HANDLE deviceHandle
- control or board device handle

Returns: M3G_SUCCESS if successful
M3G_ERROR on failure

Includes: srllib.h
m3glib.h
m3gevts.h
m3gerrs.h

Category: Utility

Mode: asynchronous

Description

The **m3g_StopTrace( )** function stops all tracing to a log file previously specified using **m3g_StartTrace( )**. This function may be called for a control device or board device only. If called on a board device, tracing is stopped for all devices opened on the board.

Parameter	Description
deviceHandle	specifies an SRL handle to a control device or board device

This function is only supported in asynchronous mode.

Termination Events

M3GEV_STOP_TRACE_CMPLT
Indicates the specified tracing level(s) has been successfully stopped for the given device(s).

M3GEV_STOP_TRACE_FAIL
Indicates the specified tracing level(s) for the given device(s) has failed to stop. The error code is included in the event.

Cautions

None.

Errors

If this function fails with M3G_ERROR, use the Standard Runtime Library (SRL) standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to obtain the error code and a pointer to the error description, respectively. Error codes are defined in *m3gerrs.h*.

For more information, see Chapter 15, “Error Codes”.

Example

```
/* Header Files: */
#include <srllib.h>
#include <m3glib.h>
#include <m3gevts.h>
#include <m3gerrs.h>
.
.
.
/* Preconditions: 1) 3G-324M Library Session has already been started.         */
/*                2) m3g_Open( ) has completed opening the board device handle */

int stopTracing(int boardDevH)
{
   if (M3G_ERROR == m3g_StopTrace(boardDevH))
   {
      log("Error:  m3g_StopTrace(%s)failed - %s\n",
           ATDV_NAMEP(boardDevH), ATDV_ERRMSGP(boardDevH));
      /* handle error… */
   }

   return SUCCESS;
} /* End of stopTracing */
.
.
.

/* SRL event handler: */
   for (;;)
   {
      if (-1 != sr_waitevt(500))
            process_event();
   }
.
.
.

void process_event(void)
{
   /* process the SRL events */
   int evType        = sr_getevttype();
   int devH          = sr_getevtdev();
   void *pSRLEvtData = sr_getevtdatap();

   switch(evType)
   {
       /*
        .
        . Other events not shown…
        .
        */

        /* Successful m3g_StopTrace termination: */
        case M3GEV_STOP_TRACE_CMPLT:
            log("M3GEV_STOP_TRACE_CMPLT for device = %s\n",
                  ATDV_NAMEP(devH));
            break;

        /* m3g_StopTrace Failure indication: */
        case M3GEV_STOP_TRACE_FAIL:
        {
            M3G_E_ERROR_TYPE* pError = (M3G_E_ERROR_TYPE*) pSRLEvtData;
            log("ERROR:  M3GEV_STOP_TRACE_FAIL for device = %s\n",
                  ATDV_NAMEP(devH));
            log("        Error value = %d\n",(int)*pError);
```

```
            /* handle error…*/
            break;
         }

         default:
            printf("Received unknown event = %d for device = %s\n",
                    evType, ATDV_NAMEP(devH));
            break;
   }
}
```

See Also

- **m3g_StartTrace( )**

Events 13

This chapter provides information about the events that may be returned by the Dialogic® 3G-324M software. Topics include:

13.1 Event Types

The Dialogic® 3G-324M software uses the following types of events:

termination events
: Termination events are returned after the completion of a function call. These events apply to the asynchronous programming model only. The 3G-324M software provides a pair of termination events for a function, to indicate successful completion or failure.

unsolicited events
: Unsolicited events are not requested by the application. They are triggered by, and provide information about, external events. Unsolicited events apply to both synchronous and asynchronous programming models.

Events are defined in the *m3gevts.h* header file.

Use **sr_waitevt( )**, **sr_enbhdlr( )** or other SRL functions to collect an event code, depending on the programming model in use. For more information, see the *Dialogic® Standard Runtime Library API Library Reference*.

13.2 Event Information

Events used by the Dialogic® 3G-324M software are described in this section. For each event, the following information is provided:

- The name of the event.
- The type of event: termination or unsolicited. See Section 13.1, "Event Types", on page 174 for more information.
- The type of device with which the event is associated: board, control, audio, or video.
- A short description explaining why the event is received and what the event indicates. Also included is whether an unsolicited event can be masked (enabled or disabled). Note that only certain unsolicited events can be masked.
- The data type, if any, of additional data that is delivered within the event.

  Use the Dialogic® Standard Runtime Library function **sr_getevtdatap( )** to retrieve the data buffer and cast it to the specified data type to process additional pertinent data. Note that the

data within this buffer must be copied or processed before the next SRL event is de-queued, at which point this buffer will be de-allocated by the SRL.

The following events are used by the Dialogic® 3G-324M software:

M3GEV_CALL_STATISTICS
Unsolicited event. Associated with control device type. Indicates the recorded statistics for the currently completed 3G call. These statistics include media and H.223 data transmission, transmission errors (CRC and malformed headers), and call duration as indicated in the attached data. This event can be masked. Data type: M3G_CALL_STATISTICS

M3GEV_CLOSE_LC_CMPLT
Termination event for **m3g_CloseLC( )**. Associated with control device type. Indicates that the specified CloseLogicalChannel or RequestChannelClose message was successfully sent to the remote 3G-324M endpoint. Data type: M3G_LOGICAL_CHANNEL_NUMBER

M3GEV_CLOSE_LC_FAIL
Termination event for **m3g_CloseLC( )**. Associated with control device type. Indicates a local failure to send the CloseLogicalChannel or RequestChannelClose request, or no response was received from the remote 3G-324M endpoint. Data type: M3G_E_ERROR_TYPE

M3GEV_DISABLE_EVENTS_CMPLT
Termination event for **m3g_DisableEvents( )**. Associated with board and control device types. Indicates that the specified events were successfully disabled.

M3GEV_DISABLE_EVENTS_FAIL
Termination event for **m3g_DisableEvents( )**. Associated with board and control device types. Indicates that the specified events failed to be disabled and remain enabled. Data type: M3G_E_ERROR_TYPE

M3GEV_ENABLE_EVENTS_CMPLT
Termination event for **m3g_EnableEvents( )**. Associated with board and control device types. Indicates that the specified events were successfully enabled.

M3GEV_ENABLE_EVENTS_FAIL
Termination event for **m3g_EnableEvents( )**. Associated with board and control device types. Indicates that the specified events failed to be enabled. Data type: M3G_E_ERROR_TYPE

M3GEV_ENDSESSION_RCVD
Unsolicited event. Associated with control device type. Indicates that an H.245 EndSession command was received from the remote 3G-324M endpoint and as a result, the current H.245 session was terminated.

M3GEV_ENDSESSION_SENT
Unsolicited event. Associated with control device type. Indicates that an H.245 EndSession command was sent to the remote 3G-324M endpoint to terminate the current H.245 session.

M3GEV_FRAMING_ESTABLISHED
Unsolicited event. Associated with control device type. Indicates that the H.223 Abstraction Layer framing and its service access points (SAPs) have been established. This event is received in one of two scenarios: (1) **m3g_StartH245( )** completed successfully, or (2) the framing layer recovered from a previous framing layer error as signaled by the M3GEV_FRAMING_LOST event.

M3GEV_FRAMING_LOST
Unsolicited event. Associated with control device type. Indicates that an error condition occurred in the framing layer which prevents the proper behavior of the H.223 multiplex layer. The H.223 framing must be restarted by calling **m3g_StopH245( )** followed by **m3g_StartH245( )**.

M3GEV_GET_LOCAL_CAPS_CMPLT
Termination event for **m3g_GetLocalCaps( )**. Associated with control, audio, and video device types. Indicates that the local capabilities for the specified device were retrieved successfully. Data type: M3G_CAPS_LIST

M3GEV_GET_LOCAL_CAPS_FAIL
Termination event for **m3g_GetLocalCaps( )**. Associated with control, audio, and video device types. Indicates that the local capabilities for the specified device failed to be retrieved. Data type: M3G_E_ERROR_TYPE

M3GEV_GET_PARM_CMPLT
Termination event for **m3g_GetParm( )**. Associated with board and control device types. Indicates that the specified parameter value was successfully retrieved as specified in the SRL event data block. Data type: M3G_PARM_INFO

M3GEV_GET_PARM_FAIL
Termination event for **m3g_GetParm( )**. Associated with board and control device types. Indicates that the specified parameter value failed to be retrieved. Data type: M3G_E_ERROR_TYPE

M3GEV_H245_MES_EVT
Unsolicited event. Associated with control device type. Indicates that an H.245 MultiplexEntrySend related transaction message was sent or received as specified in the SRL event data block. This event is only received if MultiplexEntrySend eventing is enabled via **m3g_EnableEvents( )**. This event can be masked. (Default is disabled.) Data type: M3G_MES

M3GEV_H245_MSD_EVT
Unsolicited event. Associated with control device type. Indicates that an H.245 MasterSlaveDetermination related transaction message was sent or received as specified in the SRL event data block. This event is only received if verbose MasterSlaveDetermination eventing is enabled via **m3g_EnableEvents( )**. This event can be masked. (Default is disabled.) Data type: M3G_E_MSD_EVT_TYPE

M3GEV_H245_MISC_CMD_ RCVD
Unsolicited event. Associated with control device type. Indicates that an H.245 MiscellaneousCommand indication message was received as specified in the SRL event data block. The type of MiscellaneousCommand indication message and associated parameters may be obtained from the SRL event data block. This event can be masked. (Default is enabled.) Data type: M3G_H245_MISC_CMD

M3GEV_H245_UII_RCVD
Unsolicited event. Associated with control device type. Indicates that an H.245 UII DTMF digit was received from the remote 3G-324M endpoint. This event can be masked. (Default is enabled.) Data type: M3G_H245_UII

M3GEV_IFRAME_RCVD
Unsolicited event. Associated with video device type. Indicates that an Iframe was received in-band from either the remote 3G-324M endpoint, or from the device connected to the m3g

device (IPM, MM, etc.). It only applies to H.264 Video codec. This event is only received if Iframe Report Eventing is enabled via **m3g_EnableEvents( )**. This event can be masked. (Default is disabled.) Data type: M3G_IFRAME_DATA

M3GEV_LOCAL_TCS_ACKD
Unsolicited event. Associated with control device type. Indicates that a TerminalCapabilitySet request message was successfully sent to the remote 3G-324M endpoint which acknowledged with a TerminalCapabilitySetAck response. After M3GEV_MSD_ESTABLISHED, M3GEV_REMOTE_TCS_RCVD, and M3GEV_LOCAL_TCS_ACKD events are all received, use **m3g_GetMatchedCaps( )** to retrieve common capabilities between the remote and local 3G-324M endpoints.

M3GEV_MODIFY_MEDIA_CMPLT
Termination event for **m3g_ModifyMedia( )**. Associated with audio and video device types. Indicates that the specified change to streaming between the H.223 aggregate and the specified media device successfully completed.

M3GEV_MODIFY_MEDIA_FAIL
Termination event for **m3g_ModifyMedia( )**. Associated with audio and video device types. Indicates that the specified change to streaming between the H.223 aggregate and the specified media device failed. Data type: M3G_E_ERROR_TYPE

M3GEV_MONA_PREF_MSG_RCVD
Unsolicited event. Associated with control device type. Indicates that the first MONA Preference Message was received from the remote peer indicating its transmit and receive MONA MPC capabilities. Note that an event is only dispatched on the first MONA Preference Message received from the remote peer. This event can be masked. Data type: M3G_MONA_TXRX_MPC_SUPPORT

M3GEV_MSD_ESTABLISHED
Unsolicited event. Associated with control device type. Indicates that the H.245 MasterSlaveDetermination exchange with the remote 3G-324M endpoint completed. After M3GEV_MSD_ESTABLISHED, M3GEV_REMOTE_TCS_RCVD, and M3GEV_LOCAL_TCS_ACKD events are all received, use **m3g_GetMatchedCaps( )** to retrieve common capabilities between the remote and local 3G-324M endpoints. Data type: M3G_E_H245_MSD_RESULT

M3GEV_MSD_FAILED
Unsolicited event. Associated with control device type. Indicates that the H.245 MasterSlaveDetermination exchange with the remote 3G-324M endpoint failed. The recommended next action is to re-close the H.245 session and re-start the H.245 sequence using **m3g_StopH245( )** and **m3g_StartH245( )** respectively. Data type: M3G_E_ERROR_TYPE

M3GEV_OPEN_CMPLT
Termination event for **m3g_Open( )** and **m3g_OpenEx( )**. Associated with all device types. Indicates that the specified device type successfully opened.

M3GEV_OPEN_FAIL
Termination event for **m3g_Open( )** and **m3g_OpenEx( )**. Associated with all device types. Indicates that the specified device type failed to be opened. Data type: M3G_E_ERROR_TYPE

M3GEV_OPEN_LC_CMPLT
Termination event for **m3g_OpenLC( )**. Associated with control device type. Indicates that an OpenLogicalChannelAck message was successfully received from the remote 3G-324M endpoint, acknowledging the OpenLogicalChannel request issued previously on this control device. Data type: M3G_REMOTE_OLCACK_RESP

M3GEV_OPEN_LC_FAIL
Termination event for **m3g_OpenLC( )**. Associated with control device type. Indicates that the specified capability in the OpenLogicalChannel request was not positively acknowledged from the remote 3G-324M endpoint. Data type: M3G_E_ERROR_TYPE

M3GEV_REMOTE_CLOSE_LC_RCVD
Unsolicited event. Associated with control device type. Indicates a CloseLogicalChannel or ChannelCloseRequest message was received from the remote 3G-324M endpoint. These incoming messages are always implicitly and automatically responded to via a CloseLogicalChannelAck or RequestChannelCloseAck response by the 3G-324M protocol stack. The application is only responsible for properly re-routing the associated media stream from the H.223 aggregate and stopping the media stream using **dev_PortDisconnect( )** or **dev_Disconnect( )** and **m3g_StopMedia( )**, respectively. Data type: M3G_LOGICAL_CHANNEL_NUMBER

M3GEV_REMOTE_OLC_RCVD
Unsolicited event. Associated with control device type. Indicates that an OpenLogicalChannel request message from the remote 3G-324M endpoint was received. This request must be positively acknowledged or rejected by calling **m3g_RespondToOLC( )**. Data type: M3G_REMOTE_OLC_REQ

M3GEV_REMOTE_TCS_RCVD
Unsolicited event. Associated with control device type. Indicates that a TerminalCapabilitySet request message from the remote 3G-324M endpoint was received. After M3GEV_MSD_ESTABLISHED, M3GEV_REMOTE_TCS_RCVD, and M3GEV_LOCAL_TCS_ACKD events are all received, use **m3g_GetMatchedCaps( )** to retrieve common capabilities between the remote and local 3G-324M endpoints.

M3GEV_REMOTE_VENDORID_RCVD
Unsolicited event. Associated with board device type. Indicates that an H.245 VendorIdentification message from the remote 3G-324M endpoint was received. This event can be masked. Data type: M3G_VENDORID_INFO

M3GEV_RESET_COMPLT
Termination event for **m3g_Reset( )**. Indicates that the specified devices were successfully reset or that there were no devices to recover.

M3GEV_RESET_FAIL
Termination event for **m3g_Reset( )**. Indicates that the specified devices failed to be reset. Data type: M3G_E_ERROR_TYPE

M3GEV_RESPOND_TO_LC_CMPLT
Termination event for **m3g_RespondToOLC( )**. Associated with control device type. Indicates that an OpenLogicalChannelAck or OpenLogicalChannelReject message was successfully sent to the remote 3G-324M endpoint, acknowledging the OpenLogicalChannel request issued previously on this control device. Data type: M3G_LOGICAL_CHANNEL_NUMBER

M3GEV_RESPOND_TO_LC_FAIL
Termination event for **m3g_RespondToOLC( )**. Associated with control device type. Indicates that the specified OpenLogicalChannelAck or OpenLogicalChannelReject response failed to be sent to the remote 3G-324M endpoint. Data type: M3G_E_ERROR_TYPE

M3GEV_RX_MPC_ESTABLISHED
Unsolicited event. Associated with control device type. Indicates that a Media Preconfigured Channel has been established permitting the receipt of media from the remote peer. The H.223 and media transcoding configuration is specified in the attached data. Data type: M3G_MONA_MPC

M3GEV_SEND_H245_MISC_CMD_CMPLT
Termination event for **m3g_SendH245MiscCmd( )**. Associated with control device type. Indicates that the specified H.245 MiscellaneousCommand indication message was sent successfully.

M3GEV_SEND_H245_MISC_CMD_FAIL
Termination event for **m3g_SendH245MiscCmd( )**. Associated with control device type. Indicates that the specified H.245 MiscellaneousCommand indication message failed to be sent. Data type: M3G_E_ERROR_TYPE

M3GEV_SEND_H245_UII_CMPLT
Termination event for **m3g_SendH245UII( )**. Associated with control device type. Indicates that the specified alphanumeric digit string was sent successfully.

M3GEV_SEND_H245_UII_FAIL
Termination event for **m3g_SendH245UII( )**. Associated with control device type. Indicates that the specified alphanumeric digit string failed to be sent. Data type: M3G_E_ERROR_TYPE

M3GEV_SEND_MONA_PREF_MSG
Unsolicited event. Associated with control device type. Indicates that the last MONA Preference Message was sent with specified transmit and receive MONA MPC capabilities. Note that an event will only be dispatched on the last MONA Preference Message sent to the remote peer. Data type: M3G_MONA_TXRX_MPC_SUPPORT

M3GEV_SET_PARM_CMPLT
Termination event for **m3g_SetParm( )**. Associated with board and control device types. Indicates that the specified parameter value was successfully set.

M3GEV_SET_PARM_FAIL
Termination event for **m3g_SetParm( )**. Associated with board and control device types. Indicates that the specified parameter value failed to be set. Data type: M3G_E_ERROR_TYPE

M3GEV_SET_TCS_CMPLT
Termination event for **m3g_SetTCS( )**. Associated with control device type. Indicates that the specified local capabilities were successfully set.

M3GEV_SET_TCS_FAIL
Termination event for **m3g_SetTCS( )**. Associated with control device type. Indicates that the local capabilities failed to be set on the device. Data type: M3G_E_ERROR_TYPE

M3GEV_SET_VENDORID_CMPLT
Termination event for **m3g_SetVendorId( )**. Associated with board device type. Indicates the specified vendor and product information elements have been successfully configured for the board.

M3GEV_SET_VENDORID_FAIL
Termination event for **m3g_SetVendorId( )**. Associated with board device type. Indicates the specified vendor and product information elements have failed to be configured for the board. Data type: M3G_E_ERROR_TYPE

M3GEV_START_H245_CMPLT
Termination event for **m3g_StartH245( )**. Associated with control device type. Indicates that the H.245 protocol was successfully initiated.

Note: This event does not imply any status regarding H.223 framing layer, H.245 MasterSlaveDetermination, nor H.245 TerminalCapabilitySet exchange.

M3GEV_START_H245_FAIL
Termination event for **m3g_StartH245( )**. Associated with control device type. Indicates that the H.245 protocol failed to initiate. Data type: M3G_E_ERROR_TYPE

M3GEV_START_MEDIA_CMPLT
Termination event for **m3g_StartMedia( )**. Associated with audio and video device types. Indicates streaming between the H.223 aggregate and the specified media device was successfully initiated.

M3GEV_START_MEDIA_FAIL
Termination event for **m3g_StartMedia( )**. Associated with audio and video device types. Indicates streaming between the H.223 aggregate and the specified media device failed to initiate. Data type: M3G_E_ERROR_TYPE

M3GEV_START_TRACE_CMPLT
Termination event for **m3g_StartTrace( )**. Associated with control and board device types. Indicates the specified tracing level(s) has been successfully initiated for the given device(s).

M3GEV_START_TRACE_FAIL
Termination event for **m3g_StartTrace( )**. Associated with control and board device types. Indicates the specified tracing level(s) for the given device(s) has failed to initiate. Data type: M3G_E_ERROR_TYPE

M3GEV_STOP_H245_CMPLT
Termination event for **m3g_StopH245( )**. Associated with control device type. Indicates that the H.245 protocol successfully terminated.

M3GEV_STOP_H245_FAIL
Termination event for **m3g_StopH245( )**. Associated with control device type. Indicates that the H.245 protocol failed to terminate gracefully. Data type: M3G_E_ERROR TYPE

M3GEV_STOP_MEDIA_CMPLT
Termination event for **m3g_StopMedia( )**. Associated with audio and video device types. Indicates that streaming between the H.223 aggregate and the specified media device was successfully terminated.

M3GEV_STOP_MEDIA_FAIL
Termination event for **m3g_StopMedia( )**. Associated with audio and video device types. Indicates that streaming between the H.223 aggregate and the specified media device failed to terminate. Data type: M3G_E_ERROR_TYPE

M3GEV_STOP_TRACE_CMPLT
Termination event for **m3g_StopTrace( )**. Associated with control and board device types. Indicates the specified tracing level(s) has been successfully stopped for the given device(s).

M3GEV_STOP_TRACE_FAIL
Termination event for **m3g_StopTrace( )**. Associated with control and board device types. Indicates the specified tracing level(s) for the given device(s) has failed to stop. Data type: M3G_E_ERROR_TYPE

M3GEV_TX_MPC_ESTABLISHED
Unsolicited event. Associated with control device type. Indicates that a Media Preconfigured Channel has been established permitting the sending of media from the local endpoint to the remote peer. The H.223 and media transcoding configuration is specified in the attached data. Data Type: M3G_MONA_MPC

Data Structures 14

This chapter provides an alphabetical reference to the data structures used by the Dialogic® 3G-324M software.

M3G_AMR_OPTIONS

Description

The M3G_AMR_OPTIONS structure specifies capabilities specific to the AMR-NB algorithm. This structure is a member of the M3G_AUDIO_OPTIONS structure.

Note: This structure is not currently supported.

M3G_AUDIO_CAPABILITY

```
typedef struct
{
   unsigned int       version;
   unsigned short     tableEntryNumber;
   M3G_E_DIRECTION    direction;
   M3G_E_AUDIO_TYPE   coderType;
   unsigned char      maxFramesPerSDU;
   M3G_AUDIO_OPTIONS  options;
}  M3G_AUDIO_CAPABILITY;
```

Description

The M3G_AUDIO_CAPABILITY structure specifies audio capabilities. This structure is a member of the M3G_CAPABILITY data structure.

Field Descriptions

The fields of the M3G_AUDIO_CAPABILITY data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

direction
: direction of specified audio in perspective of local endpoint. Valid values are:
  - M3G_E_IDLE – no streaming
  - M3G_E_TX – transmit from local to remote
  - M3G_E_RX – receive from remote to local
  - M3G_E_TXRX – bi-directional streaming

  For the local 3G-324M endpoint, only M3G_E_TX and M3G_E_RX may be used in the terminal capability settings in **m3g_SetTCS( )** as asymmetric media (audio and video) transcoding is supported. The remote 3G-324M endpoint, however, may specify symmetric media capabilities (M3G_E_TXRX) in its TerminalCapabilitySet message.

tableEntryNumber
: table entry number of capability within CapabilityTableEntry of H.245 TerminalCapabilitySet message. Read-only field provided for information. This field is not used in OpenLogicalChannel requests.

coderType
: type of audio codec. Valid values are:
  - M3G_E_G7231 – G.723.1 transcoding
  - M3G_E_GSM_AMR_NB – GSM adaptive multi-rate narrow band (AMR-NB) codec

maxFramesPerSDU
: maximum number of audio frames per AL-SDU. Valid range is 1 - 256. Default value returned in **m3g_GetLocalCaps( )** is 2 for G.723.1 and 1 for AMR.

options
: M3G_AUDIO_OPTIONS union specifying additional elements unique to the supported codec algorithms.

M3G_AUDIO_OPTIONS

```
typedef union
{
      M3G_G7231_OPTIONS   g7231;
      M3G_AMR_OPTIONS     amr;
} M3G_AUDIO_OPTIONS;
```

Description

The M3G_AUDIO_OPTIONS union specifies elements unique to the supported audio codec algorithms. This union is a member of the M3G_AUDIO_CAPABILITY structure.

Field Descriptions

The fields of the M3G_AUDIO_OPTIONS union are described as follows:

g7231
: structure specifying capabilities specific to G.723.1. See the M3G_G7231_OPTIONS structure for more information.

amr
: structure specifying capabilities specific to AMR-NB. See the M3G_AMR_OPTIONS structure for more information.

M3G_CALL_STATISTICS

```
typedef struct
{
   unsigned int callDuration;
   unsigned int rxMediaCrcErrs;
   unsigned int rxAudioPacketErrs;
   unsigned int rxVideoPacketErrs;
   unsigned int rxTotalBytes;
   unsigned int rxMediaBytes;
   unsigned int rxAudioBytes;
   unsigned int rxVideoBytes;
   unsigned int rxStuffingBytes;
   unsigned int rxMediaPackets;
   unsigned int rxAudioPackets;
   unsigned int rxVideoPackets;
   unsigned int rxMuxPdus;
   unsigned int rxMuxPduBytes;
   unsigned int txTotalBytes;
   unsigned int txMediaBytes;
   unsigned int txAudioBytes;
   unsigned int txVideoBytes;
   unsigned int txStuffingBytes;
   unsigned int txMediaPackets;
   unsigned int txAudioPackets;
   unsigned int txVideoPackets;
   unsigned int txMuxPdus;
   unsigned int txMuxPduBytes;
} M3G_CALL_STATISTICS;
```

Description

The M3G_CALL_STATISTICS structure provides call statistics on quality of service related items such as media and H.223 data transmission, errors, and call duration. This information is specified in the M3GEV_CALL_STATISTICS event.

Field Descriptions

The fields of the M3G_CALL_STATISTICS data structure are described as follows:

callDuration
duration of call in milliseconds

rxMediaCrcErrs
number of CRC errors detected in media packets received by the demultiplexer

rxAudioPacketErrs
number of errors detected in audio packets received by the demultiplexer

rxVideoPacketErrs
number of errors detected in video packets received by the demultiplexer

rxTotalBytes
total number of bytes received by the demultiplexer

rxMediaBytes
number of bytes dedicated to media minus any adaptation layer (AL) overhead received by the demultiplexer

rxAudioBytes
number of bytes dedicated to audio received by the demultiplexer

rxVideoBytes
number of bytes dedicated to video received by the demultiplexer

rxStuffingBytes
number of bytes recognized as stuffing received by the demultiplexer

rxMediaPackets
number of packets dedicated to media received by the demultiplexer

rxAudioPackets
number of packets dedicated to audio received by the demultiplexer

rxVideoPackets
number of packets dedicated to video received by the demultiplexer

rxMuxPdus
number of MUX-PDUs received by the demultiplexer

rxMuxPduBytes
number of bytes in MUX-PDUs received by the demultiplexer

txTotalBytes
total number of bytes transmitted by the multiplexer

txMediaBytes
number of bytes dedicated to media minus any adaptation layer (AL) overhead transmitted by the multiplexer

txAudioBytes
number of bytes dedicated to audio transmitted by the multiplexer

txVideoBytes
number of bytes dedicated to video transmitted by the multiplexer

txStuffingBytes
number of stuffing bytes transmitted by the multiplexer

txMediaPackets
number of packets dedicated to media transmitted by the multiplexer

txAudioPackets
number of packets dedicated to audio transmitted by the multiplexer

txVideoPackets
number of packets dedicated to video transmitted by the multiplexer

txMuxPdus
number of MUX-PDUs transmitted by the multiplexer

txMuxPduBytes
number of bytes in MUX-PDUs transmitted by the multiplexer

M3G_CAPABILITY

```
typedef union
{
     M3G_H223_CAPABILITY       h223Capability;
     M3G_AUDIO_CAPABILITY      audioCapability;
     M3G_VIDEO_CAPABILITY      videoCapability;
} M3G_CAPABILITY;
```

Description

The M3G_CAPABILITY union specifies H.223 multiplex, audio or video capabilities. An array of this union is contained in the M3G_CAPS_LIST structure.

Field Descriptions

The fields of the M3G_CAPABILITY union are described as follows:

h223Capability
M3G_H223_CAPABILITY structure that specifies the H.223 multiplex capabilities

audioCapability
M3G_AUDIO_CAPABILITY structure that specifies the audio capabilities

videoCapability
M3G_VIDEO_CAPABILITY structure that specifies the video capabilities

M3G_CAPS_LIST

```
typedef struct
{
      unsigned int       version;
      unsigned short     numCaps;
      M3G_E_CAPABILITY   capabilityType;
      M3G_CAPABILITY     capability[MAX_CAPABILITIES_PER_DEVTYPE];
} M3G_CAPS_LIST;
```

Description

The M3G_CAPS_LIST structure specifies capabilities. This structure is used by **m3g_GetLocalCaps( )** and **m3g_SetTCS( )** to indicate capabilities for the local endpoint. This structure is used by **m3g_GetMatchedCaps( )** to indicate capabilities common between local and remote endpoints.

The audio and video capabilities in the M3G_CAPS_LIST array are listed in decreasing order of preference. The most preferred capability is ordered first in the array, while the least preferred is ordered last.

Field Descriptions

The fields of the M3G_CAPS_LIST data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

numCaps
: number of elements in the capability array up to a maximum of twenty

capabilityType
: data type contained in the M3G_CAPABILITY union. Valid values are:
  - M3G_E_H223_CAPABILITY – H.223 multiplex capability type
  - M3G_E_AUDIO_CAPABILITY – audio capability type
  - M3G_E_VIDEO_CAPABILITY – video capability type

capability
: array of numCaps M3G_CAPABILITY data types specifying H.223, audio or video capabilities

Example

For an example of this data structure, see the Example section for the **m3g_GetLocalCaps( )**, **m3g_SetTCS( )**, and **m3g_GetMatchedCaps( )** functions.

M3G_FASTUPDATE_GOB

```
typedef structure
{
   unsigned int                    numFirstGOB;
   unsigned int                    numGOBs;
} M3G_FASTUPDATE_GOB;
```

Description

The M3G_FASTUPDATE_GOB structure specifies information transmitted or received within an H.245 MiscellaneousCommand indication message type of videoFastUpdateGOB to or from the remote 3G-324M endpoint.

This structure is a member of the M3G_H245_MISC_CMD_PARAMS union.

Field Descriptions

The fields of the M3G_FASTUPDATE_GOB data structure are described as follows:

numFirstGOB
number of first Group of Blocks (GOB) in H.263 video stream to update. Valid range is 0 – 17.

numGOBs
number of GOBs in H.263 video stream to update. Valid range is 1 – 18.

Example

For an example of this data structure, see the Example section for the **m3g_SendH245MiscCmd( )** function.

M3G_FASTUPDATE_MB

```
typedef structure
{
   unsigned int                    numFirstGOB;
   unsigned int                    numFirstMB;
   unsigned int                    numMBs;
} M3G_FASTUPDATE_MB;
```

Description

The M3G_FASTUPDATE_MB structure specifies information transmitted or received within an H.245 MiscellaneousCommand indication message type of videoFastUpdateMB to or from the remote 3G-324M endpoint.

This structure is a member of the M3G_H245_MISC_CMD_PARAMS union.

Field Descriptions

The fields of the M3G_FASTUPDATE_MB data structure are described as follows:

numFirstGOB
: number of first group of blocks (GOB) in H.263 video stream to update. Valid range is 0 – 255.

numFirstMB
: number of first macro block (MB) in H.263 video stream to update. Valid range is 0 – 8192.

numMBs
: number of MBs in H.263 video stream to update. Valid range is 1 – 8192.

Example

For an example of this data structure, see the Example section for the **m3g_SendH245MiscCmd( )** function.

M3G_G7231_OPTIONS

```
typedef struct
{
      M3G_BOOL         silenceSup;
} M3G_G7231_OPTIONS;
```

Description

The M3G_G7231_OPTIONS structure specifies capabilities specific to the G.723.1 algorithm. This structure is a member of the M3G_AUDIO_OPTIONS structure.

Field Descriptions

The field of the M3G_G7231_OPTIONS structure is described as follows:

silenceSup
boolean value specifying whether silence suppression is supported

M3G_H221_NONSTD

```
typedef struct
{
   unsigned char t35CountryCode;
   unsigned char t35Extension;
   unsigned short manufacturerCode;
} M3G_H221_NONSTD;
```

Description

The M3G_H221_NONSTD structure is used to encode H.221 identifiers.

Field Descriptions

The fields of the M3G_H221_NONSTD data structure are described as follows:

t35CountryCode
: octets as defined in Annex A/T.35

t35Extension
: octet assigned nationally, unless t35CountryCode is 0xFF, in which case the second octet contains the country code according to Annex B/T.35

manufacturerCode
: two octets assigned nationally

M3G_H223_CAPABILITY

```
typedef struct
{
     unsigned int        version;
     unsigned short      adaptationLayerMedia;
     unsigned short      ALxM_AnnexC_Media;
     unsigned short      maxAL2SDUSize;
     unsigned short      maxAL3SDUSize;
     M3G_BOOL            frameH223AnnexA;
     M3G_BOOL            frameH223DoubleFlag;
     M3G_BOOL            frameAnnexB;
     M3G_BOOL            frameAnnexBWithHead;
     unsigned short      maxAL1MPDUSize;
     unsigned short      maxAL2MPDUSize;
     unsigned short      maxAL3MPDUSize;
     M3G_BOOL            rsCodeCapability;
     M3G_BOOL            mobileOpXmitCap;
     unsigned char       bitRate;
} M3G_H223_CAPABILITY;
```

Description

The M3G_H223_CAPABILITY structure specifies H.223 multiplex capabilities. This structure is a member of the M3G_CAPABILITY data structure.

Field Descriptions

The fields of the M3G_H223_CAPABILITY data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

adaptationLayerMedia
: bitmask used to indicate adaptation layers supported per audio and video. Valid values are:

- M3G_AUDIO_AL1
- M3G_AUDIO_AL2
- M3G_AUDIO_AL3
- M3G_VIDEO_AL1
- M3G_VIDEO_AL2
- M3G_VIDEO_AL3

Default value returned in **m3g_GetLocalCaps( )** is M3G_AUDIO_AL2 | M3G_VIDEO_AL3.

ALxM_AnnexC_Media
: bitmask used to indicate H.223 Annex C adoption layer media support. Valid values are:

- M3G_NO_ANNEXC
- M3G_AUDIO_AL1M
- M3G_AUDIO_AL2M
- M3G_AUDIO_AL3M
- M3G_VIDEO_AL1M

- M3G_VIDEO_AL2M
- M3G_VIDEO_AL3M

Default value returned in **m3g_GetLocalCaps( )** is M3G_NO_ANNEXC.

maxAL2SDUSize
: maximum AL2 SDU size. Valid range is 0 - 65535. Default value returned in **m3g_GetLocalCaps( )** is 200.

maxAL3SDUSize
: maximum AL3 SDU size. Valid range is 0 - 65535. Default value returned in **m3g_GetLocalCaps( )** is 200.

frameH223AnnexA
: boolean used to indicate support for Annex A framing. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

frameH223DoubleFlag
: boolean used to indicate support for double flags in framing. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

frameAnnexB
: boolean used to indicate support for Annex B framing. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

frameAnnexBWithHead
: boolean used to indicate support for Annex B framing with optional headers. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

maxAL1MPDUSize
: maximum Annex C AL1M PDU size. Valid range is 0 - 65535. Default value returned in **m3g_GetLocalCaps( )** is 0.

maxAL2MPDUSize
: maximum Annex C AL2M PDU size. Valid range is 0 - 65535. Default value returned in **m3g_GetLocalCaps( )** is 0.

maxAL3MPDUSize
: maximum Annex C AL3M PDU size. Valid range is 0 - 65535. Default value returned in **m3g_GetLocalCaps( )** is 0.

rsCodeCapability
: boolean used to indicate support for receipt of Annex C Reed-Solomon encoded PDUs. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

mobileOpXmitCap
: boolean used to indicate whether the mobileOperationTransmitCapability element is encoded in the H223Capability inside Terminal Capability Set messages. The H.245 mobileOperationTransmitCapability element settings are specified by the frameH223AnnexA, frameH223DoubleFlag, frameAnnexB, and frameAnnexBWithHead structure elements. Default value returned in **m3g_GetLocalCaps( )** is M3G_TRUE.

bitRate
: value identifying the unframed bitrate to transmit the output of the H.223 multiplex. Supported values are 320 and 640 (in units of 100 bps). Default value is 640.

M3G_H223_LC_PARAMS

```
typedef structure
{
   unsigned int                    version;
   M3G_E_ADAPTATION_LYR_TYPE       adaptationLayerType;
   M3G_BOOL                        segmentable;
   unsigned char                   AL3_ControlFieldSize;
   unsigned int                    AL3_SendBufferSize;
   M3G_E_ALxM_HEADER_TYPE          ALxM_HeaderFormat;
   M3G_BOOL                        ALxM_ALPDUInterleaving;
   M3G_E_ALxM_CRC_TYPE             ALxM_CRCType;
   M3G_E_ADAPTATION_LYR_ARQ_TYPE   ALxM_ARQType;
   unsigned char                   ALxM_ARQMaxNumRetrans;
   unsigned int                    ALxM_ARQSendBufferSize;
   M3G_BOOL                        AL1M_SplitSDU;
   unsigned char                   ALxM_RCPCCodeRate;
} M3G_H223_LC_PARAMS;
```

Description

The M3G_H223_LC_PARAMS structure specifies H.223 multiplex parameters to be encoded in the OpenLogicalChannel message. This structure is used by **m3g_OpenLC( )**.

Use the INIT_M3G_H223_LC_PARAMS( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_H223_LC_PARAMS data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

adaptationLayerType
: type of adaptation layer to use. The data type is an enumeration that defines the following values:
  - M3G_E_AL1_FRAMED – Adaptation Layer 1 framed transfer mode
  - M3G_E_AL1_UNFRAMED – Adaptation Layer 1 unframed transfer mode
  - M3G_E_AL2_WITH_SEQ_NUMS – Adaptation Layer 2 with sequence numbers
  - M3G_E_AL2_WITHOUT_SEQ_NUMS – Adaptation Layer 2 without sequence numbers
  - M3G_E_AL3 – Adaptation Layer 3
  - M3G_E_AL1M_FRAMED – AL1M (H.223 Annex C/D) framed transfer mode. Not currently supported.
  - M3G_E_AL1M_UNFRAMED – AL1M (H.223 Annex C/D) unframed transfer mode. Not currently supported.
  - M3G_E_AL2M – AL2M (H.223 Annex C/D). Not currently supported.
  - M3G_E_AL3M – AL3M (H.223 Annex C/D). Not currently supported.

segmentable
: boolean indicating if channel is designated to be segmentable. Audio channels must be configured as non-segmentable (M3G_FALSE) while video channels must be configured as segmentable (M3G_TRUE).

AL3_ControlFieldSize
size of AL3 control field in octets. Valid range is 0 – 2.
- 0 – not present; valid only in non-AL3 adaptation layers
- 1 – control field size of one octet
- 2 – control field size of two octets

AL3_SendBufferSize
size of AL3 send buffer in octets. Valid range is 0 – 16777215.
- 0 – not present; valid only in non-AL3 adaptation layers
- > 0 – send buffer size; must be specified for AL3 adaptation layers

ALxM_HeaderFormat
AL1M, AL2M or AL3M header format. The data type is an enumeration that defines the following values:
- M3G_E_ALx_HEADER_SEBCH16 – Systematic Extended Bose-Chaudhuri-Hocquenghem (16, 5) header format
- M3G_E_ALx_HEADER_GOLAY24 – EGolay (24, 12) header format. Ignored in non-ALxM adaptation layer types.

AlxM_ALPDUInterleaving
boolean indicating whether channel supports AL PDU interleaving for AL1M, AL2M and AL3M. Ignored in non-ALxM adaptation layer types.

ALxM_CRCType
AL1M and AL3M Cyclic Redundancy Check (CRC) lengths. The data type is an enumeration that defines the following values:
- M3G_E_AL_CRC_4 – AL1M and AL3M 4 bit CRC
- M3G_E_AL_CRC_12 – AL1M and AL3M 2 bit CRC
- M3G_E_AL_CRC_20 – AL1M and AL3M 20 bit CRC
- M3G_E_AL_CRC_28 – AL1M and AL3M 28 bit CRC
- M3G_E_AL_CRC_8 – AL1M and AL3M 8 bit CRC
- M3G_E_AL_CRC_16 – AL1M and AL3M 16 bit CRC
- M3G_E_AL_CRC_32 – AL1M and AL3M 32 bit CRC
- M3G_E_AL_CRC_NONE – no CRC.

Field is ignored if adaptation layer is not AL1M or AL3M.

ALxM_ARQType
AL1M and AL3M Automatic Repeat Request (ARQ) mode. The data type is an enumeration that defines the following values:
- M3G_E_AL_ARQ_NONE – FEC_ONLY mode
- M3G_E_AL_ARQ_TYPEI_FINITE – ARQ type I mode with finite retransmissions
- M3G_E_AL_ARQ_TYPEI_INFINITE – ARQ type I mode with infinite retransmisssions
- M3G_E_AL_ARQ_TYPEII_FINITE – ARQ type II mode with finite retransmissions
- M3G_E_AL_ARQ_TYPEII_INFINITE – ARQ type II mode with infinite retransmisssions. Ignored if adaptation layer is not AL1M or AL3M. Not supported by H.223 Annex D.

ALxM_ARQMaxNumRetrans
maximum number of ARQ retransmissions for ARQ Type I and Type II finite types. Valid range is 0 – 16. Ignored if configuration is not ARQ type I or type II finite retransmission for AL1M or AL3M.

ALxM_ARQSendBufferSize
: size of ARQ send buffer in octets. Valid range is 0 to 166777215. Ignored if configuration is not ARQ type I or type II for AL1M or AL3M.

AL1M_SplitSDU
: boolean used to indicate support for SDU splitting for AL1M. Ignored if adaptation layer is not AL1M. Not supported by H.223 Annex D.

ALxM_RCPCCodeRate
: H.223 Annex C Rate Compatible Punctured Convolutional Code Rate for AL1M and AL3M. Value is expressed in units of 8/*n*, where *n* is 8 to 32. Ignored if adaptation layer is not AL1M or AL3M.

M3G_H223_SESSION

```
typedef struct
{
   unsigned int             version;
   M3G_E_H223_MUX_LVL_TYPE  defaultH223MuxLevel;
   unsigned int             maxALSDUSize;
   M3G_BOOL                 isWNSRPEnabled;
   M3G_BOOL                 isMultipleMsgsPerPdu;
   M3G_BOOL                 isMONAEnabled;
} M3G_H223_SESSION;
```

Description

The M3G_H223_SESSION structure specifies the default configuration of the H.223 multiplex layer. This structure is used by **m3g_StartH245( )**.

Use the INIT_M3G_H223_SESSION( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_H223_SESSION data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

defaultH223MuxLevel
: Specifies whether the H.223 multiplex level is initiated at level 0, level 1, level 2 or level 3.

maxALSDUSize
: Specifies the maximum size of the Abstraction Layer Service Data Unit (AL-SDU) in octets.

isWNSRPEnabled
: Boolean specifying whether Windowed NSRP control frame signaling is supported.

isMultipleMsgsPerPdu
: Boolean specifying whether multiple messages may be sent per PDU.

isMONAEnabled
: Boolean specifying whether Media Oriented Negotiation Acceleration (MONA) support is enabled for the 3G call. If enabled, Media Preconfigured Channels (MPC) and Accelerated Connection Procedure (ACP) are supported for the call. Supported MPCs signaled within MONA Preference Messages are assigned per the terminal capability set specified in **m3g_SetTCS( )**. If the default terminal capabilities are used (preferred method), the default MPC-TX bits are 1, 4, 5 to signify AMR, MPEG-4 and H.263 transmit capabilities, respectively; the default MPC-RX bits are 1 and 5 to signify AMR and H.263 receive capabilities, respectively.

Example

For an example of this data structure, see the Example section for the **m3g_StartH245( )** function.

M3G_H245_MISC_CMD

```
typedef structure
{
   unsigned int                version;
   M3G_LOGICAL_CHANNEL_NUMBER  logicalChannelNumber;
   M3G_E_H245_MISC_CMD_TYPE    h245MiscCmdType;
   M3G_MISC_CMD_PARAMS         h245MiscCmdParams;
} M3G_H245_MISC_CMD;
```

Description

The M3G_H245_MISC_CMD structure specifies information transmitted or received within an H.245 MiscellaneousCommand indication message to or from the remote 3G-324M endpoint. This structure is encoded within the **m3g_SendH245MiscCmd( )** function and received via the M3GEV_H245_MISC_CMD_RCVD event.

Use the INIT_M3G_H245_MISC_CMD( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_H245_MISC_CMD data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logicalChannelNumber
: number of H.245 logical channel

h245MiscCmdType
: type of H.245 MiscellaneousCommand. The data type is an enumeration that defines the following values:
  - M3G_E_FAST_UPDATE_PICTURE – send/received videoFastUpdatePicture command to enter fast-update mode.
  - M3G_E_FAST_UPDATE_GOB – send/received videoFastUpdateGOB command to perform fast-update on one or more groups of blocks (GOB) as specified in the fastUpdateGOB structure within h245MiscCmdParams union.
  - M3G_E_FAST_UPDATE_MB – send/received videoFastUpdateMB command to perform fast-update on one or more macro blocks (MB) as specified in the fastUpdateMB structure within h245MiscCmdParams union.
  - M3G_E_TEMP_SPAT_TRDFF – send/received videoTemporalSpatialTradeoff command to set the trade-off between spatial and temporal resolution to the value as specified in the tempSpatialTrdff structure within h245MiscCmdParams union.
  - M3G_E_VIDEO_FREEZE – send/received videoFreezePicture command to freeze the current picture once complete.
  - M3G_E_SYNC_EVERY_GOB – send/received videoSendSyncEveryGOB command to use synchronization for every GOB until a videoSendSyncEveryGOBCancel is received.
  - M3G_E_NOSYNC_EVERY_GOB – send/received videoSendSyncEveryGOBCancel command to re-determine the frequency of GOB synchronizations.

h245MiscCmdParams
union that specifies parameters of H.245 MiscellaneousCommand. See M3G_H245_MISC_CMD_PARAMS for more information.

Example

For an example of this data structure, see the Example section for the **m3g_SendH245MiscCmd( )** function.

M3G_H245_MISC_CMD_PARAMS

```
typedef union
{
   void*                  noParams;
   M3G_FASTUPDATE_GOB     fastUpdateGOB;
   M3G_FASTUPDATE_MB      fastUpdateMB;
   M3G_TEMPSPTRDFF        tempSpatialTrdff;
} M3G_H245_MISC_CMD_PARAMS;
```

Description

The M3G_H245_MISC_CMD_PARAMS union specifies parameters of H.245 MiscellaneousCommand. This union is a member of the M3G_H245_MISC_CMD structure.

Field Descriptions

The fields of the M3G_H245_MISC_CMD_PARAMS union are described as follows:

fastUpdateGOB
: specifies parameter information for videoFastUpdateGOB command. See M3G_FASTUPDATE_GOB structure for more information.

fastUpdateMB
: specifies parameter information for videoFastUpdateMB command. See M3G_FASTUPDATE_MB structure for more information.

tempSpatialTrdff
: specifies parameter information for videoTemporalSpatialTradeoff command. See M3G_TEMPSPTRDFF for more information.

M3G_H245_UII

```
typedef structure
{
   unsigned int                  version;
   unsigned short                numDigits;
   unsigned char                 digitBuffer[MAX_NUM_DIGITS];
} M3G_H245_UII;
```

Description

The M3G_H245_UII structure is encoded within the **m3g_SendH245UII( )** function or within the M3GEV_H245_UII_RCVD event. This structure specifies DTMF digits transmitted or received in an H.245 UserInputIndication message to or from the remote 3G-324M endpoint.

Use the INIT_M3G_H245_UII( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_H245_UII data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

numDigits
: number of digits specified in the digit buffer

digitBuffer
: DTMF digits to be transmitted or that were received via H.245 UserInputIndication message(s)

Example

For an example of this data structure, see the Example section for the **m3g_SendH245UII( )** function.

M3G_H263_OPTIONS

```
typedef struct
{
   unsigned short      bppMaxKb;
   unsigned char       sqcifMPI;
   unsigned char       qcifMPI;
   M3G_BOOL            unrestrictedVector;
   M3G_BOOL            arithmeticCoding;
   M3G_BOOL            advancedPrediction;
   M3G_BOOL            pbFrames;
   M3G_BOOL            temporalSpatialTradeoffCap;
   M3G_BOOL            errorCompensation;
} M3G_H263_OPTIONS;
```

Description

The M3G_H263_OPTIONS structure specifies capabilities specific to the H.263 algorithm. This structure is a member of the M3G_VIDEO_OPTIONS union.

Field Descriptions

The fields of the M3G_H263_OPTIONS structure are described as follows:

bppMaxKb
: maximum number of bits per encoded picture, measured in units of 1024 bits, that the receiver may decode. Valid range is 64 – 65535 or 0 (not present). Default value returned in **m3g_GetLocalCaps( )** is 0.

sqcifMPI
: Sub Quarter Common Intermediate Format minimum picture interval in units of 1/29.97. Valid range is 0 – 32 where 0 is no capability. Supported values are: 2, 3, and 5 which represent 15, 10 and 6 frames/sec respectively. Default value returned in **m3g_GetLocalCaps( )** is 2.

Note: Only one H.263 format may be specified inside an OpenLogicalChannel request. Thus when formatting an H.263 videoCapability for use in **m3g_OpenLC( )**, only sqcifMPI or qcifMPI may be non-zero, but not both.

qcifMPI
: Quarter Common Intermediate Format minimum picture interval in units of 1/29.97. Valid range is 0 – 32 where 0 is no capability. Supported values are: 2, 3, and 5 which represent 15, 10 and 6 frames/sec respectively. Default value returned in **m3g_GetLocalCaps( )** is 2.

Note: Only one H.263 format may be specified inside an OpenLogicalChannel request. Thus when formatting an H.263 videoCapability for use in **m3g_OpenLC( )**, only sqcifMPI or qcifMPI may be non-zero, but not both.

unrestrictedVector
: boolean used to indicate support of unrestricted motion vector of H.263 Annex D. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

arithmeticCoding
: boolean used to indicate support of syntax-based arithmetic encoding. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

advancedPrediction
: boolean used to indicate support of syntax-based advance prediction of H.263 Annex F. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

pbFrames
: boolean used to indicate support of interleaving P and B frames. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

temporalSpatialTradeoffCap
: boolean used to indicate support of capability to trade off between temporal and spatial resolution. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

errorCompensation
: boolean used to indicate support of error compensation as defined in H.263 Appendix I. Default value returned in **m3g_GetLocalCaps( )** is M3G_FALSE.

M3G_H264_OPTIONS

```
typedef struct
{
   unsigned char  profile;                                  /* H264 Profile */
   unsigned char  level;                                    /* H264 Level   */
   unsigned char  decoderConfigLength;                      /* H264 DCI length */
   unsigned char  decoderConfigInfo[OCTET_STRING_SIZE];     /* used in local and  remote H.245
                                                             * OLC  May be present
                                                             * exactly once for Logical
                                                             * Channel Signalling  */
   unsigned char    h264SignalingMask;                      /* used to signal parameters
                                                             * described below in local and
                                                             * remote H.245 TCS - optional */
   unsigned char       profileIOP;                          /* used in local and  remote
                                                             * H.245 TCS - optional */
   } M3G_H264_OPTIONS;

   /* Bits in h264SignalingMask are assigned as follows:*/
   #define M3G_H264_ACCEPT_REDUNDANT_SLICES  0X01|
   #define M3G_H264_NAL_ALIGNED_MODE  0X02
```

Description

The M3G_H264_OPTIONS structure specifies capabilities specific to the H.264 algorithm. This structure is a member of the M3G_VIDEO_OPTIONS union.

Field Descriptions

The fields of the M3G_H264_OPTIONS structure are described as follows:

profile
: Mandatory H.264 capability as specified in ITU-T Rec. H.241. Latest version of 3GPP TS 26.111 mandates H.264 Baseline profile for 3G-324M. Default value returned in **m3g_GetLocalCaps( )** is 64 (Baseline Profile).

level
: Mandatory H.264 capability as specified in ITU-T Rec. H.241. Latest version of 3GPP TS 26.111 mandates H.264 Level 1 (value 15) for 3G-324M. Default value returned in **m3g_GetLocalCaps( )** is 15 (Level 1).

decoderConfigLength
: Length of decoderConfigurationInformation octet string. This element is only used in encoding or decoding of H.245 OpenLogicalChannel requests. Note that 3GPP TS 26.111 does not mandate that a DCI shall be specified within a video OpenLogicalChannel request. Default value returned in **m3g_GetLocalCaps( )** is 0.

decoderConfigInfo
: Optionally used in OpenLogicalChannel requests to specify the configuration of the decoder or a particular object (bitstream). Format is specified in 3GPP TS 26.111. Default value returned in **m3g_GetLocalCaps( )** is '\0'. When MONA MPC is used to establish a Video H.264 channel, the following DCI octet string is reported: "00 00 00 01 27 42 a0 0a 95 a0 b1 3a 01 e1 10 8d 40 00 00 00 01 28 ce 06 6a".

h264SignalingMask

- Mask 0x01 indicates the capability to use H.264 redundant slices. Format is specified in 3GPP TS 26.111. Default value returned in **m3g_GetLocalCaps( )** is false.
- Mask 0x02 indicates that every AL-SDU carrying H.264 shall contain an integer number of NAL units and that the start of the AL-SDU shall be aligned with the start of a NAL. Format is specified in 3GPP TS 26.111. Default value returned in **m3g_GetLocalCaps( )** is false.

profileIOP

Indicates that the capability to decode H.264 streams is limited to a common subset of the algorithmic features included in the indicated profile and level. Format is specified in 3GPP TS 26.111. Default value returned in **m3g_GetLocalCaps( )** is 0.

M3G_IFRAME_DATA

```
typedef struct
{
   M3G_E_DIRECTION  direction;                 /* Direction the Iframe was received from */
   unsigned int     iframeSize                 /* size of the Iframe */
   M3G_BOOL         dciChanged;                /* Indicates whether the DCI changed */
   unsigned char    decoderConfigLength;       /* H.264 DCI length */
   unsigned char    decoderConfigInfo[OCTET_STRING_SIZE];   /* inband H264 DCI*/
} M3G_IFRAME_DATA;
```

Description

This data structure specifies inband information from either the remote 3G-324M endpoint, or from the device connected to the M3G device (IPM, MM, etc.). This data structure is received via the M3GEV_IFRAME_RCVD event. It only applies to the H.264 video codec.

Field Descriptions

The fields of the M3G_IFRAME_DATA structure are described as follows:

direction
: Direction from which the Iframe was received. Possible values:
  - M3G_E_TX: from the device connected to the M3G device
  - M3G_E_RX: from the remote 3G-324M endpoint

iframeSize
: Size of the Iframe in bytes.

dciChanged
: Indicates if the inband DCI has changed according to the application expectation. It means that the inband DCI either changed initially versus the out-of-band DCI, or changed mid-stream.

decoderConfigLength
: H.264 DCI length. If no DCI is attached to the reported Iframe, the length is 0.

decoderConfigInfo
: Optional H.264 DCI Data, if decoderConfigLength is different from 0.

M3G_MONA_MPC

```
typedef struct
{
   unsigned int version;
   M3G_LOGICAL_CHANNEL_NUMBER logicalChannelNumber;
   M3G_H223_LC_PARAMS h223MultiplexParams;
   M3G_E_CAPABILITY capabilityType;
   M3G_CAPABILITY mediaCapability;
} M3G_MONA_MPC;
```

Description

The M3G_MONA_MPC structure provides information on the newly established MONA media preconfigured channel (MPC) including the logical channel number, H.223, and media transcoding configuration. This information is included in the M3GEV_TX_MPC_ESTABLISHED event and the M3GEV_RX_MPC_ESTABLISHED event.

Field Descriptions

The fields of the M3G_MONA_MPC data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logicalChannelNumber
: logical channel number associated with the media preconfigured channel

h223MultiplexParams
: H.223 multiplex parameters to use for this channel

capabilityType
: media capability type of the logical channel being acknowledged. The data type is an enumeration that defines the following values:
  - M3G_E_AUDIO_CAPABILITY – audio capability type
  - M3G_E_VIDEO_CAPABILITY – video capability type

mediaCapability
: capability describing the established media preconfigured channel

M3G_MONA_TXRX_MPC_SUPPORT

```
typedef struct
{
   unsigned short rxMPCMask;
   unsigned short txMPCMask;
} M3G_MONA_TXRX_MPC_SUPPORT;
```

Description

The M3G_MONA_TXRX_MPC_SUPPORT data structure specifies the Media Oriented Negotiation Acceleration Procedure (MONA) Media Preconfigured Channel Transmit and Receive Bits (MPC-TX and MPC-RX) as defined by Table K.5/H.324. This data is delivered in the M3GEV_SEND_MONA_PREF_MSG event and the M3GEV_MONA_PREF_MSG_RCVD event.

Field Descriptions

The fields of the M3G_MONA_TXRX_MPC_SUPPORT data structure are described as follows:

rxMPCMask
: low order 13 bits of mask used to describe which MPC configurations the MONA terminal is capable of **receiving** as per Table K.5/H.324

txMPCMask
: low order 13 bits of mask used to describe which MPC configurations the MONA terminal is capable of **sending** as per Table K.5/H.324

Bits are assigned as follows as per Table K.15/H.324:

- M3G_MPC_AMR_BIT
- M3G_MPC_AMR_WB_BIT
- M3G_MPC_H264_BIT
- M3G_MPC_MPEG4_BIT
- M3G_MPC_H263_BIT
- M3G_MPC_RESERVED_BIT6
- M3G_MPC_RESERVED_BIT7
- M3G_MPC_RESERVED_BIT8
- M3G_MPC_RESERVED_BIT9
- M3G_MPC_RESERVED_BIT10
- M3G_MPC_RESERVED_BIT11
- M3G_MPC_OPERATOR_BIT12
- M3G_MPC_OPERATOR_BIT13

M3G_MPEG4_OPTIONS

```
typedef struct
{
   unsigned char  profileAndLevel;
   unsigned char  object;
   unsigned char  decoderConfigLength;
   unsigned char  decoderConfigInfo[OCTET_STRING_SIZE];   /* used in local and
                                                           * remote H.245 OLC
                                                           * Request messages only;
                                                           * ignored in Terminal
                                                           * Capability Set messages
                                                           */
   M3G_BOOL       visualBackChannel;
} M3G_MPEG4_OPTIONS;
```

Description

The M3G_MPEG4_OPTIONS structure specifies capabilities specific to the MPEG-4 algorithm. This structure is a member of the M3G_VIDEO_OPTIONS union.

Field Descriptions

The fields of the M3G_MPEG4_OPTIONS structure are described as follows:

profileAndLevel
: process the particular profiles in combination with the level as given in Table G.1, “FLC table for profile_and_level_indication” of the ISO/IEC 14496-2 standard. Default value returned in **m3g_GetLocalCaps( )** is 1.

object
: set of tools to be used by the decoder of the bitstream contained in the logical channel as given in Table 6-10, “FLC table for video_object_type indication” of the ISO/IEC 14496-2 standard. Default value returned in **m3g_GetLocalCaps( )** is 1.

decoderConfigLength
: length of decoderConfigurationInformation octet string. This element is only used in encoding or decoding of H.245 OpenLogicalChannel requests.

decoderConfigInfo
: optionally used in OpenLogicalChannel requests to specify the configuration of the decoder for a particular object (bitstream). (See subclause 6.2.1, “Start Codes” and subclauses K.3.1, “VideoObject” to K.3.4, “FaceObject” of the ISO/IEC 14496-2 standard.) If no octet string is specified in **m3g_OpenLC( )** for an MPEG-4 logical channel, the default decoderConfigurationInformation octet string will be specified as “00-00-01-b0-08-00-00-01-b5-09-00-00-01-00-00-00-01-20-00-84-5d-4c-28-2c-20-90-a2-8f”.

visualBackChannel
: boolean indicating the transmitter receives backward channel messages or the receiver sends backward channel messages that are provided in ISO/IEC 14496-2. Default is M3G_FALSE.

M3G_NONSTANDARD_ID

```
typedef struct
{
   M3G_E_NONSTD_TYPE oidType;
   union
   {
      M3G_OBJECT_ID oid;
      M3G_H221_NONSTD h221NonStd;
   } oidValue;
} M3G_NONSTANDARD_ID;
```

Description

The M3G_NONSTANDARD_ID structure is used to encode vendor information in the H.245 vendorIdentification indication message in the format of a non-standard identifier as defined by ITU-T Recommendation H.245.

Field Descriptions

The fields of the M3G_NONSTANDARD_ID data structure are described as follows:

oidType
identifier format:
- M3G_E_OID_TYPE – object identifier
- M3G_E_H221_ID_TYPE – H.221 type of identifier
- M3G_E_NONSTD_TYPE – non-standard type of identifier

oidValue
nested union defining the identifier

M3G_OBJECT_ID

```
typedef struct
{
   unsigned char length;
   unsigned char objectId[OBJECTID_SIZE];
} M3G_OBJECT_ID;
```

Description

The M3G_OBJECT_ID structure is used to encode ASN.1 object identifiers.

Field Descriptions

The fields of the M3G_OBJECT_ID data structure are described as follows:

length
: length of the object identifier in octets

objectId
: array of octets of specified length. Note that this differs from a C-style string as zero values do not represent the NULL terminator.

M3G_OCTET_STRING

```
typedef struct
{
      unsigned char             length;
      unsigned char             octet[OCTET_STRING_SIZE];
} M3G_OCTET_STRING;
```

Description

The M3G_OCTET_STRING structure represents an ASN.1 OCTET STRING type. This structure is a member of the M3G_PARM_INFO data structure.

Field Descriptions

The fields of the M3G_OCTET_STRING data structure are described as follows:

length
: length of ASN.1 OCTET STRING in octets up to a maximum of 255.

octet
: array of octets composing ASN.1 OCTET STRING. Currently only used to specify decoderConfigurationInformation octet strings for MPEG-4 transcoding.

M3G_PARM_INFO

```
typedef struct
{
   unsigned int       version;
   M3G_E_PRM_TYPE     parameterType;
   union
   {
       M3G_BOOL                          boolParam;
       M3G_H245_TERMINAL_TYPE            h245TerminalType;
       M3G_MAX_CCSRL_SEGMENT_SIZE        maxCCSRLSegmentSize;
       M3G_AMR_PAYLOAD_FORMAT            amrPayloadFormat;
       M3G_BITMASK                       bitmask; /* for internal use only */
       M3G_OCTET_STRING                  octetString;
       M3G_SKEW_ADJUSTMENT               skewAdjustment;
       M3G_VIDEO_BIT_RATE                videoBitRate;
       M3G_VIDEO_FRAME_RATE              videoFrameRate;
       M3G_VFU_TIMER_VAL                 vfuTimer;
       M3G_H223_SYNC_TIMER_VAL           vfuSyncTimer;
       M3G_IFRAME_NOTIFY_CONTROL_MASK    iframeNotifyControlMask;
   }   parmValue;
} M3G_PARM_INFO;
```

Description

The M3G_PARM_INFO structure contains parameters used to configure a device. The structure is used by the **m3g_SetParm( )** and **m3g_GetParm( )** functions.

Parameters may be specified for a board device, a control device, or both types of devices. Setting one or more parameters on a board device sets the default values for all control devices associated with that board. Not all parameters may be set on both board and control devices; for example, M3G_E_PRM_AMR_PAYLOAD_FORMAT can be set on a board device only. See Table 3 for details.

Use the INIT_M3G_PARM_INFO( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_PARM_INFO data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

parameterType
: data type contained in the parmValue union. See Table 3.

parmValue
: union specifying parameter values. See Table 3.

Parameter type, description, and parameter values are described in Table 3, in alphabetical order by parameter type.

Table 3. M3G_PARM_INFO Parameter Types and Parameter Values

Parameter Type	Data Type	Description/Values
M3G_E_PRM_AMR_ PAYLOAD_FORMAT	M3G_AMR_PAYLOAD_ FORMAT	Supported on board device only. Adaptive multi-rate codec payload format. Valid values: • AMR_PAYLOAD_BW_EFFICIENT – bandwidth efficient mode • AMR_PAYLOAD_OCTET_ALIGNED – octet aligned mode
M3G_E_PRM_ AUDIOVISUALSYNC	M3G_BOOL	Supported on board device and control device. Enables audio and video synchronization when both media streams are present in an H.223 multiplex. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_AUTO_VFU_ PERIOD	M3G_VFU_TIMER_VAL	Supported on board device and control device. When an incoming video logical channel is first opened, immediately and automatically send an H.245 videoFastUpdatePicture Miscellaneous Command message to the remote encoder. The parameter value specifies the timer interval in seconds for subsequent periodic generation of the videoFastUpdatePicture command or zero for disabled. The default value is zero.
M3G_E_PRM_EARLY_MES	M3G_BOOL	Supported on board device and control device. When opening logical channel using standard H.245 procedures, send MES messages prior to receiving positive acknowledgement of the OLCAck to minimize media transmission delay. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_H223_SYNC_ TIMER	M3G_H223_SYNC_ TIMER_VAL	Supported on board device and control device. Specifies the timer value in milliseconds for the H.223 synchronization timer for which the H.223 multiplexer must synchronize upon flags received within the incoming bitstream. The default value is 5000 msec.
M3G_E_PRM_H245_ TERMINAL_TYPE	M3G_H245_TERMINAL _TYPE	Supported on board device and control device. The value of H.245 terminal types is used in the H.245 MasterSlaveDetermination procedure. The terminal type values are compared and the terminal with the larger terminal type number is determined to be the master. If the terminal type numbers are the same, the statusDeterminationNumbers, which are randomly set internally, are compared using modulo arithmetic to determine which terminal is the master. The default value is 50.
M3G_E_PRM_H264_TX_ DCI	M3G_OCTET_STRING	Supported on control device only. Used to specify the transmitted out-of-band decoderConfigurationInformation octet string used from PStream. The associated parmValue union element is processed as an octetString. See the M3G_OCTET_STRING structure, which represents an ASN.1 OCTET STRING type.
M3G_E_PRM_IFRAME_ NOTIFY_CONTROL_MASK	M3G_IFRAME_NOTIFY _CONTROL_MASK	Controls the H.264 Iframe notification event. Valid bit field constants are: • IFRAME_NOTIFY_RX: Notify any Iframe events received from 3G. • IFRAME_NOTIFY_TX: Notify any Iframe events received from PSTREAM. • IFRAME_NOTIFY_DCI_CHANGE_RX: Notify only Iframes which DCI changed from 3G. • IFRAME_NOTIFY_DCI_CHANGE_TX: Notify only Iframes which DCI changed from PSTREAM.

Parameter Type	Data Type	Description/Values
M3G_E_PRM_MAX_ CCSLR_SEGMENT	M3G_MAX_CCSLR_ SEGMENT	Supported on board device and control device. Maximum size in octets of a control channel segmentation and reassembly layer (CCSRL) segment to allow. The default value is 255.
M3G_E_PRM_MPEG4_ TX_DCI	M3G_OCTET_STRING	Supported on control device only. Used in gateway deployments to specify the transmitted decoderConfigurationInformation octet string used within MPEG-4 transcoding after an MPEG-4 logical channel has been established. The associated parmValue union element is processed as an octetString. See the M3G_OCTET_STRING structure, which represents an ASN.1 OCTET STRING type.
M3G_E_PRM_MPEG4_ RX_DCI	M3G_OCTET_STRING	Supported on control device only. Used in gateway deployments to specify the received decoderConfigurationInformation octet string used within MPEG-4 transcoding after an MPEG-4 logical channel has been established. The associated parmValue union element is processed as an octetString. See the M3G_OCTET_STRING structure, which represents an ASN.1 OCTET STRING type.
M3G_E_PRM_RELAY_ DIGIT_TO_MEDIA_DEV	M3G_BOOL	Supported on board device and control device. Relay H.245 UII digits from remote 3G-324M endpoint to local media devices as proprietary RFC 2833 encoding. Valid values: • M3G_FALSE [default] – false • M3G_TRUE – true
M3G_E_PRM_RELAY_ DIGIT_TO_H245UII	M3G_BOOL	Supported on board device and control device. Relay DTMF digits detected in an audio stream from local media devices to remote 3G-324M endpoint via H.245 UII message. Valid values: • M3G_FALSE [default] – false • M3G_TRUE – true
M3G_E_PRM_RELAY_ FASTUPDATE_TO_H245	M3G_BOOL	Supported on board device and control device. Relay proprietary RFC 2833 encoded videoFastUpdate message from local media devices to remote 3G-324M endpoint as an H.245 MiscellaneousCommand. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_RELAY_ FASTUPDATE_TO_ MEDIA_DEV	M3G_BOOL	Supported on board device and control device. Relay H.245 MiscellaneousCommand type videoFastUpdate message from remote 3G-324M endpoint to local media devices as proprietary RFC 2833 encoding. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_RELAY_ TEMPORALSPATIALTRADE OFF_TO_MEDIA_DEV	M3G_BOOL	Supported on board device and control device. Relay H.245 MiscellaneousCommand of type videoTemporalSpatialTradeoff message from remote 3G-324M endpoint to local media devices as proprietary RFC 2833 encoding. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_RELAY_ TEMPORALSPATIALTRADE OFF_TO_H245	M3G_BOOL	Supported on board device and control device. Relay proprietary RFC 2833 encoded videoTemporalSpatialTradeoff message from local media devices to remote 3G-324M endpoint as an H.245 MiscellaneousCommand. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true

Parameter Type	Data Type	Description/Values
M3G_E_PRM_RELAY_VIDEOFREEZE_TO_MEDIA_DEV	M3G_BOOL	Supported on board device and control device. Relay H.245 MiscellaneousCommand of type videoFreezePicture from remote 3G-324M endpoint to local media devices as proprietary RFC 2833 encoding. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_RELAY_VIDEOFREEZE_TO_H245	M3G_BOOL	Supported on board device and control device. Relay proprietary RFC 2833 encoded videoFreezePicture message from local media devices to remote 3G-324M endpoint as an H.245 MiscellaneousCommand. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_RETRANSMIT_ON_IDLE	M3G_BOOL	Supported on board device and control device. Send retransmissions of Numbered Simple Retransmission Protocol (NSRP) and Windowed NSRP (WNSRP) commands for H.245 messages during the call setup stages only. Valid values: • M3G_FALSE [default] – false • M3G_TRUE – true
M3G_E_PRM_RX_SKEW_ADJUSTMENT	M3G_SKEW_ADJUSTMENT	Supported on board device and control device. Received audio and visual (AV) skew adjustment. This offset is added to adjust the AV delivery from nominal. This value is also added to any AV skew added by the remote endpoint via the H.245 Skew Indication message. A positive value delays the audio stream. A negative value delays the video stream. Valid range is 500 to -500 in units of milliseconds. **Note:** The application is responsible for resetting this value back to nominal at the end of the call.
M3G_E_PRM_SKEWINDICATION	M3G_BOOL	Supported on board device and control device. Relay H.245 h223SkewIndication messages as proprietary RFC 2833 encoding between (both to and from) the H.223 multiplex and local media devices. Valid values: • M3G_FALSE – false • M3G_TRUE [default] – true
M3G_E_PRM_TX_SKEW_ADJUSTMENT	M3G_SKEW_ADJUSTMENT	Supported on board device and control device. Transmitted audio and visual (AV) skew adjustment. This offset is added to adjust the AV delivery from nominal. A positive value delays the audio stream. A negative values delays the video stream. Valid range is 500 to -500 in units of milliseconds. **Note:** The application is responsible for resetting this value back to nominal at the end of the call.
M3G_E_PRM_VIDEO_BIT_RATE	M3G_VIDEO_BIT_RATE	Supported on board device and control device. Video coder bit rate. Valid range is 20000 to 54000 in units of bits per second. Default value is 40000. **Note:** The application is responsible for resetting this value back to nominal at the end of the call.
M3G_E_PRM_VIDEO_FRAME_RATE	M3G_VIDEO_FRAME_RATE	Supported on board device and control device. Video coder frame rate in units of frames per second. Valid values: • VIDEO_FRAME_RATE_6_FPS (default) – 6 frames per second (fps) • VIDEO_FRAME_RATE_10_FPS – 10 fps • VIDEO_FRAME_RATE_15_FPS – 15 fps

Example

For an example of this data structure, see the Example section for the **m3g_SetParm( )** function.

M3G_REMOTE_CLOSED_LC

```
typedef struct
{
   unsigned in  t              version;
   M3G_LOGICAL_CHANNEL_NUMBER  logicalChannelNumber;
   M3G_E_CHAN_CLOSE_REASON     reason;
} M3G_REMOTE_CLOSED_LC;
```

Description

The M3G_REMOTE_CLOSED_LC structure is encoded within the M3GEV_REMOTE_CLOSE_LC_RCVD event to indicate that the remote 3G-324M endpoint requested the closure of a logical channel.

Field Descriptions

The fields of the M3G_REMOTE_CLOSED_LC data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logicalChannelNumber
: number of H.245 logical channel requested by remote 3G-324M endpoint to close

reason
: H.245 reason provided in the CloseLogicalChannel message or the RequestChannelClose message:
  - M3G_E_REQ_CHAN_CLOSE_UNKNOWN – unknown
  - M3G_E_REQ_CHAN_CLOSE_NORMAL – normal
  - M3G_E_REQ_CHAN_CLOSE_REOPEN – reopen
  - M3G_E_REQ_CHAN_CLOSE_RESERV_FAIL – reservationFailure

M3G_REMOTE_OLC_REQ

```
typedef struct
{
   unsigned int                 version;
   M3G_LOGICAL_CHANNEL_NUMBER   logicalChannelNumber;
   M3G_H223_LC_PARAMS           h223MultiplexParams;
   M3G_E_CAPABILITY             capabilityType;
   M3G_CAPABILITY               mediaCapability;
} M3G_REMOTE_OLC_REQ;
```

Description

The M3G_REMOTE_OLC_REQ structure is encoded within the M3GEV_REMOTE_OLC_RCVD event. The structure specifies information received in an H.245 OpenLogicalChannel request from the remote 3G-324M endpoint.

Field Descriptions

The fields of the M3G_REMOTE_OLC_REQ data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logicalChannelNumber
: number of H.245 reverse logical channel requested by remote endpoint to open

h223MultiplexParams
: H.223 multiplex parameters to use for this channel. See M3G_H223_LC_PARAMS for more information.

capabilityType
: media capability type of the logical channel being requested. The data type is an enumeration that defines the following values:
  - M3G_E_AUDIO_CAPABILITY – audio capability type
  - M3G_E_VIDEO_CAPABILITY – video capability type

mediaCapability
: media capability being requested to open in reverse channel. See M3G_CAPABILITY for more information.

Example

For an example of this data structure, see the Example section for **m3g_RespondToOLC( )**.

M3G_REMOTE_OLCACK_RESP

```
typedef struct
{
   unsigned int                version;
   M3G_LOGICAL_CHANNEL_NUMBER  logicalChannelNumber;
   M3G_E_CAPABILITY            capabilityType;
} M3G_REMOTE_OLCACK_RESP;
```

Description

The M3G_REMOTE_OLCACK_RESP structure is encoded within the M3GEV_OPEN_LC_CMPLT event. The structure specifies information from the OpenLogicalChannelAck response received from the remote 3G-324M endpoint.

Field Descriptions

The fields of the M3G_REMOTE_OLCACK_RESP data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logicalChannelNumber
: number of H.245 forward logical channel acknowledged by remote endpoint

capabilityType
: the media capability type of the logical channel being acknowledged. The data type is an enumeration that defines the following values:
  - M3G_E_AUDIO_CAPABILITY – audio capability type
  - M3G_E_VIDEO_CAPABILITY – video capability type

Example

For an example of this data structure, see the Example section for **m3g_StartMedia( )**.

M3G_SIMULTANEOUS_CAP_SET

```
typedef struct
{
   M3G_CAPS_LIST *    pH223Capabilities;
   M3G_CAPS_LIST *    pAudioCapabilities;
   M3G_CAPS_LIST *    pVideoCapabilities;
} M3G_SIMULTANEOUS_CAP_SET;
```

Description

The M3G_SIMULTANEOUS_CAP_SET structure specifies the default local set of terminal capabilities. This structure is used by the **m3g_SetTCS( )** function.

Field Descriptions

The fields of the M3G_SIMULTANEOUS_CAP_SET data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

pH223Capabilities
: pointer to an array of M3G_CAPS_LIST containing H.223 multiplex capabilities

pAudioCapabilities
: pointer to an array of M3G_CAPS_LIST containing audio capabilities

pVideoCapabilities
: pointer to an array of M3G_CAPS_LIST containing video capabilities

Example

For an example of this data structure, see the Example section for **m3g_SetTCS( )**.

M3G_START_STRUCT

```
typedef struct
{
      unsigned int   version;
      unsigned short numVirtBoards;
      unsigned short numEndpoints;
} M3G_START_STRUCT;
```

Description

The M3G_START_STRUCT structure contains configuration settings used by the **m3g_Start( )** function to instantiate the 3G-324M library.

It is recommended that you use the INIT_M3G_START_STRUCT macro, in the *m3glib.h* header file, to initialize the structure. You can then override any of the default values initialized by the macro before calling **m3g_Start( )**.

Field Descriptions

The fields of the M3G_START_STRUCT data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

numVirtBoards
: number of virtual boards that the 3G-324M library instantiates

numEndpoints
: number of 3G-324M endpoints or channels that the 3G-324M library instantiates on each virtual board.

  To instantiate the maximum number of 3G-324M endpoints licensed, set to 0. When set to 0, the number of licensed 3G-324M endpoints that are instantiated by the library will be returned in numEndpoints upon successful completion of **m3g_Start( )**.

Example

For an example of this data structure, see the Example section for **m3g_Start( )**.

M3G_TEMPSPTRDFF

```
typedef unsigned int M3G_TEMPSPTRDFF;
```

Description

The M3G_TEMPSPTRDFF is a scalar typedef. It indicates to the receiving video decoder the current trade-off between temporal and spatial resolution. A value of 0 indicates a high spatial resolution and a value of 31 indicates a high frame rate. The values from 0 to 31 indicate monotonically a higher frame rate.

This typedef is a member of the M3G_H245_MISC_CMD_PARAMS union.

M3G_TRACE_INFO

```
typedef struct
{
   unsigned int       version;
   const char *       logfile;
   unsigned int *     bitmask;
} M3G_TRACE_INFO;
```

Description

The M3G_TRACE_INFO structure specifies configuration information for 3G-324M tracing for a device or devices. This structure is used by the **m3g_StartTrace( )** function.

Use the INIT_M3G_TRACE_INFO( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_TRACE_INFO data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

logfile
: null-terminated C-style character string specifying log file name to be opened for the given device or devices. If a zero length string or empty string (null terminator only) is specified, the logging defaults to the system logger within /var/log/messages. The pointer to the character string must not be NULL. If a file name without a full path is specified, the default location for log files is /usr/dialogic/data.

bitmask
: bitmask to configure tracing for a given device or devices:
  - M3G_TRACE_H245 – record all H.245 messages
  - M3G_TRACE_H223 – record both transmit and receive H.223 multiplexed bitstreams
  - M3G_TRACE_AUDIO – record audio bitstreams multiplexed and demultiplexed
  - M3G_TRACE_VIDEO – record video bitstreams multiplexed and demultiplexed
  - M3G_TRACE_INTERNALS – enable internal 3G-324M module debug tracing showing parameters, call flow, and functional call processing
  - M3G_TRACE_STATISTICS – record 3G-324M session statistics

Example

For an example of this data structure, see the Example section for **m3g_StartTrace( )**.

M3G_VENDORID_INFO

```
typedef struct
{
   unsigned int     version;
   M3G_NONSTANDARD_ID vendor;
   M3G_OCTET_STRING productNumber;
   M3G_OCTET_STRING versionNumber;
} M3G_VENDORID_INFO;
```

Description

The M3G_VENDORID_INFO structure specifies information transmitted and received within an H.245 vendorIdentification indication message to and from the remote 3G-324M endpoint. This structure is used by the **m3g_SetVendorId( )** function.

Use the INIT_M3G_VENDORID_INFO( ) inline function to initialize the structure.

Field Descriptions

The fields of the M3G_VENDORID_INFO data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

vendor
: equipment manufacturer sourcing the vendorIdentification message encoded as an H.245 nonstandard identifier object

productNumber
: product number of the equipment sourcing the H.245 vendorIdentification message encoded within an ASN.1 octet string

versionNumber
: version number of the product sourcing the H.245 vendorIdentification message encoded within an ASN.1 octet string

Example

For an example of this data structure, see the Example section for **m3g_StartTrace( )**.

M3G_VIDEO_CAPABILITY

```
typedef struct
{
   unsigned int       version;
   unsigned short     tableEntryNumber;
   M3G_E_DIRECTION    direction;
   M3G_E_VIDEO_TYPE   coderType;
   unsigned int       maxBitRate;
   M3G_VIDEO_OPTIONS  options;
}  M3G_VIDEO_CAPABILITY;
```

Description

The M3G_VIDEO_CAPABILITY structure specifies video capabilities. This structure is a member of the M3G_CAPABILITY union.

Field Descriptions

The fields of the M3G_VIDEO_CAPABILITY data structure are described as follows:

version
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Set to the symbolic constant M3G_LIBRARY_VERSION which defines the current version of the library.

tableEntryNumber
: table entry number of capability within CapabilityTableEntry of H.245 TerminalCapabilitySet message. Read-only field provided for information. This field is not used in OpenLogicalChannel requests.

direction
: direction of specified video from the perspective of the local endpoint. The data type is an enumeration that defines the following values:
  - M3G_E_IDLE – no streaming
  - M3G_E_TX – transmit from local to remote
  - M3G_E_RX – receive from remote to local
  - M3G_E_TXRX – bi-directional streaming

  For the local 3G-324M endpoint, only M3G_E_TX and M3G_E_RX may be used in the terminal capability settings in **m3g_SetTCS( )** as asymmetric media (audio and video) transcoding is supported. The remote 3G-324M endpoint, however, may specify symmetric media capabilities (M3G_E_TXRX) in its TerminalCapabilitySet message.

coderType
: type of video codec. The data type is an enumeration that defines the following values:
  - M3G_E_H263 – H.263 codec
  - M3G_E_MPEG4 – MPEG-4 codec
  - M3G_E_H264 – H.264 codec

maxBitRate
: maximum bit rate between 1 – 9200. Default value returned in **m3g_GetLocalCaps( )** is 560.

options
: union specifying the additional elements unique to the supported codec algorithms. See M3G_VIDEO_OPTIONS for more information.

Example

For an example of this data structure, see the Example section for **m3g_GetLocalCaps( )**.

M3G_VIDEO_OPTIONS

```
typedef union
{
      M3G_H263_OPTIONS   h263;
      M3G_MPEG4_OPTIONS  mpeg4;
      M3G_H264_OPTIONS   h264;
} M3G_VIDEO_OPTIONS;
```

Description

The M3G_VIDEO_OPTIONS union specifies elements unique to the supported video codec algorithms. This union is a member of the M3G_VIDEO_CAPABILITY structure.

Field Descriptions

The fields of the M3G_VIDEO_OPTIONS union are described as follows:

h263
: structure that specifies capabilities specific to the H.263 algorithm. See M3G_H263_OPTIONS structure for more information.

mpeg4
: structure that specifies capabilities specific to the MPEG-4 algorithm. See M3G_MPEG4_OPTIONS structure for more information.

h264
: structure that specifies capabilities specific to the H.264 algorithm. See M3G_H264_OPTIONS structure for more information.

Error Codes 15

This chapter describes the error codes used in the Dialogic® 3G-324M API library.

Errors are defined in *m3gerrs.h*.

The following error codes may be returned either by the 3G-324M library function or through error codes included in M3GEV_ failure events. For more information on events, see Chapter 13, "Events".

M3G_E_ERR_BUSY
Device is busy

M3G_E_ERR_ERR_TIMEOUT
Timer expired while pending on a transaction response

M3G_E_ERR_IN_STREAM_OVFLOW
Input stream overflow

M3G_E_ERR_IN_STREAM_UNDRUN
Input stream underrun

M3G_E_ERR_INTERNAL
Internal error

M3G_E_ERR_INV_ARGUMENT_VALUE
Argument value is invalid

M3G_E_ERR_INV_MODE
Invalid mode argument

M3G_E_ERR_INV_PARM_ID
Device does not support this parameter ID

M3G_E_ERR_INV_STATE
Invalid state to execute this function

M3G_E_ERR_INVALID_CAPS_FOR_DEVICE
Specified capabilities cannot be support on this device

M3G_E_ERR_INVALID_DEVICE
Supplied device handle is invalid

M3G_E_ERR_LIB_NOT_STARTED
3G-324M library has not been started

M3G_E_ERR_NO_MATCH_FOUND
No match or set intersection could be found in the specified capabilities

M3G_E_ERR_NO_MEM
Library cannot obtain the memory to perform this operation

M3G_E_ERR_O_STREAM_ OVFLOW
Output stream overflow

M3G_E_ERR_O_STREAM_UNDRUN
Output stream underrun

M3G_E_ERR_OLC_REJ_DATA_TYPE_COMB_NOT_ALWD
Received OpenLogicalChannelReject response with cause multicastChannelNotAllowed

M3G_E_ERR_OLC_REJ_DATA_TYPE_COMB_NOT_SUP
Received OpenLogicalChannelReject response with cause dataTypeALCombinationNotSupported

M3G_E_ERR_OLC_REJ_DATA_TYPE_NOT_AVAILABLE
Received OpenLogicalChannelReject response with cause dataTypeNotAvailable

M3G_E_ERR_OLC_REJ_DATA_TYPE_NOT_SUPPORTED
Received OpenLogicalChannelReject response with cause dataTypeNotSupported

M3G_E_ERR_OLC_REJ_INSUFF_BW
Received OpenLogicalChannelReject response with cause insufficientBandwdith

M3G_E_ERR_OLC_REJ_INV_DEP_CHANNEL
Received OpenLogicalChannelReject response with cause invalidDependentChannel

M3G_E_ERR_OLC_REJ_INVALID_SESSIONID
Received OpenLogicalChannelReject response with cause invalidSessionID

M3G_E_ERR_OLC_REJ_M_S_CONFLICT
Received OpenLogicalChannelReject response with cause masterSlaveConflict

M3G_E_ERR_OLC_REJ_REPLCMT_FOR_REJECTED
Received OpenLogicalChannelReject response with cause replacementForRejected

M3G_E_ERR_OLC_REJ_SEP_STCK_EST_FAILED
Received OpenLogicalChannelReject response with cause separateStackEstablishmentFailed

M3G_E_ERR_OLC_REJ_UNKOWN_DATA_TYPE
Received OpenLogicalChannelReject response with cause unknownDataType

M3G_E_ERR_OLC_REJ_UNS_REV_PARMS
Received OpenLogicalChannelReject response with cause unsuitableReverseParameters

M3G_E_ERR_OLC_REJ_UNSPECIFIED
Received OpenLogicalChannelReject response with cause unspecified

M3G_E_ERR_OLC_REJ_WAIT_FOR_COMM_MODE
Received OpenLogicalChannelReject response with cause waitForCommunicationMode

M3G_E_ERR_PHYSICAL_LAYER
Error in physical layer must be resolved

M3G_E_ERR_PROTOCOL
Protocol error

M3G_E_ERR_STREAM_OPEN_ERR
Error in opening stream

M3G_E_ERR_TCS_REJ_DESC_CAP_EXCEEDED
Received TerminalCapabilitySetReject response with cause descriptorCapacityExceeded

M3G_E_ERR_TCS_REJ_TBL_ENT_CAP_EXCEEDED
Received TerminalCapabilitySetReject response with cause tableEntryCapacityExceeded

M3G_E_ERR_TCS_REJ_UND_TBL_ENTRY_USED
Received TerminalCapabilitySetReject response with cause undefinedTableEntryUsed

M3G_E_ERR_TCS_REJ_UNSPECIFIED
Received TerminalCapabilitySetReject response with cause unspecified

M3G_E_KERN_MEM
Kernel memory error

M3G_E_NO_ERROR
Function completed successfully with no error

M3G_E_NO_RESOURCE
No resources are available

M3G_E_REJECTED_BY_PEER
Requested transaction is rejected by peer

M3G_E_UNSUPPORTED
Requested action is unsupported

Glossary

3GPP: 3rd Generation Partnership Project. A cooperation of international standards bodies for the development of technical specifications for cellular systems that support voice and high-speed data, known as third-generation (3G) systems. Established in 1998, 3GPP comprises North American, European, Japanese, Korean, and Chinese standards development organizations.

3G-324M: Based on ITU-T H.324 recommendation modified by 3GPP for purposes of 3GPP circuit switched network based video telephony.

ACP: Accelerated Connect Procedure

H.223 Annex A: ITU-T recommendation covering multiplexing protocol for low bit rate multimedia mobile communication over low error-prone channels.

H.223 Annex B: ITU-T recommendation covering multiplexing protocol for low bit rate multimedia mobile communication over moderate error-prone channels.

H.223 Annex C: ITU-T recommendation covering multiplexing protocol for low bit rate multimedia mobile communication over highly error-prone channels.

H.223 Annex D: ITU-T recommendation covering optional multiplexing protocol for low bit rate multimedia mobile communication over highly error-prone channels.

H.245: ITU-T recommendation covering control protocol for multimedia communication.

H.324: ITU-T recommendation for low bit rate circuit-switched multimedia service in 3GPP networks.

H.324 Annex K: ITU-T recommendation adding support for Media Oriented Negotiation Acceleration (MONA).

Media Oriented Negotiation Acceleration (MONA): A group of complementary standards designed to significantly reduce delay in H.324 call setup time. The standards include Accelerated Connect Procedure (ACP), Media Preconfigured Channels (MPC), and Signaling Preconfigured Channel (SPC).

MPC: Media Preconfigured Channels

MSD: Master Slave Determination

OLC: Open Logical Channel

SPC: Signaling Preconfigured Channel

TCS: Terminal Capabilities Set