Dialogic®

3G-324M API

Library Reference

April 2008

05-2558-004

Publication Date: April 2008

Document Number: 05-2558-004

Contents

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2558-004	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_MISC_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.
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.

Document No.	Publication Date	Description of Revisions
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
- Applicability
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This publication provides a reference to the functions, events, data structures, and error codes in the 3G-324M API library.

Applicability

This document is published for Dialogic® Multimedia Software for AdvancedTCA Release 1.1 and Release 2.0.

This document may also be applicable to other software releases (including service updates) on Linux or Windows® operating systems. 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 you are familiar with the Linux or Windows® operating system and have experience using the C programming language.

How to Use This Publication

The information in this document is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the various categories of 3G-324M functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to the 3G-324M functions.
- Chapter 3, "Events" provides an alphabetical reference to events that may be returned by the 3G-324M software.
- Chapter 4, "Data Structures" provides an alphabetical reference to the 3G-324M data structures.
- Chapter 5, "Error Codes" presents a list of error codes that may be returned by the 3G-324M software.

Related Information

Refer to the following sources for more information:

- *Dialogic® Standard Runtime Library API Programming Guide*
- *Dialogic® Standard Runtime Library API Library Reference*
- *Dialogic® Device Management API Library Reference*
- *Dialogic® IP Media Library API Programming Guide*
- *Dialogic® IP Media Library API Library Reference*
- 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 compatibility issues, restrictions and limitations, known problems, 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*

Function Summary by Category 1

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

1.1 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

1.2 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

1.3 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

1.4 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 2

This chapter provides an alphabetical reference to the functions in the 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.

2.1 Function Syntax Conventions

The 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 5, “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 3, “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 5, “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)

To change default settings, use **m3g_EnableEvents( )** and **m3g_DisableEvents**( ). See Chapter 3, "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 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 5, “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)

To change default settings, use **m3g_EnableEvents( )** and **m3g_DisableEvents( )**. See Chapter 3, "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 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

■ 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 5, “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 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 3, “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 5, “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 5, “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 1, “M3G_PARM_INFO Parameter Types and Parameter Values”, on page 153.

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 3, “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 5, “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 5, “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.

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 3, “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 5, “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:

control device

The control device provides functional interfaces for:

- 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 a narrow band interface user plane (Nb UP). 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 format of the **deviceName** string to specify a control device is “m3gB*m*T*n*”, where “*m*” is the specified board number and “*n*” is the specified channel number.

board device

The board device is used to set global default values. The board device handle is used in 3G-324M function calls such as **m3g_SetParm( )** and **m3g_EnableEvents( )** to specify parameter values for all applicable control, audio, and video device instances subsequently opened on the specified board.

The format of the **deviceName** string to specify a board device is “m3gB*m*”, where “*m*” is the specified board number.

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

audio device

The audio device represents the audio connections 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 format of the **deviceName** string to specify an audio device 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 connections 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 format of the **deviceName** string to specify a video device 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 3, "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 5, "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 3, “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 5, “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 3, “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 5, “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;

           /* 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;
        }

        /* 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 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++)
   {
       /* Capability 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 */
```

■ See Also

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

m3g_Reset( )

Name:	int m3g_Reset(deviceHandle)	
Inputs:	SRL_DEVICE_HANDLE deviceHandle	• SRL handle to the virtual board device
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. This function should only be used to recover devices that were not properly closed due to an abnormal or improper shutdown of some process, and should not be used otherwise.

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 3, “Events”.

■ Cautions

This function should only be used to recover previously opened devices that were not closed due to an abnormal shutdown of a process. 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. The function returns the M3GEV_RESET_COMPLT event if it successfully recovered one or more devices, or if there were no devices to recover.

■ 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 5, “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 abstraction 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 3, “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 5, "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 3, “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 5, “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 3, “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 5, "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 1, "M3G_PARM_INFO Parameter Types and Parameter Values", on page 153.

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 3, "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 5, "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 3, “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 5, “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.

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 5, "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

Only call this function on a control device type after all associated audio and video media devices allocated to their respective H.223 multiplex channels have 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 3, “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 5, “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 NbUP 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                                   */
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 */
   }

   /* 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)
   {
      /* 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;
   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)
   {
      /* Start opening appropriate logical channels */
      startOpeningLogicalChannels(pMyDev);
   }
   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 or reverse logical channels have been successfully opened between the local and the remote 3G-324M endpoints. Also, ensure that all audio and/or video devices have been configured with the appropriate capability settings as negotiated per the respective H.245 OLCAck.

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 3, "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 5, "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 logfile. 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.

A post-processing logfile 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 logfile is created for every channel for each category that has been enabled in the specified logfile. 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 5, “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 5, “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 all associated H.245 logical channel usage is complete and all logical channels have been closed. 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 3, "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 5, "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 3, “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 5, "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 5, “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 3

This chapter provides information about the events that may be returned by the 3G-324M software. Topics include:

3.1 Event Types

The 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*.

3.2 Event Information

Events used by the 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 3.1, “Event Types”, on page 121 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 3G-324M software:

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_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_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. Data type: M3G_VENDOR_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_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_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

Data Structures 4

This chapter provides an alphabetical reference to the data structures used by the 3G-324M software. The following data structures are described:

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_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.

■ 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_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;
     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 – 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 – 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_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.

■ 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_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_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;
   }   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 1 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 1.

parmValue
: union specifying parameter values. See Table 1.

Parameter type, description, and parameter values are described in Table 1, in alphabetical order by parameter type.

Table 1. 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_AUDIOVIS UALSYNC	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_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_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. 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

Parameter Type	Data Type	Description/Values
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_ TEMPORALSPATIALTRA DEOFF_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_ TEMPORALSPATIALTRA DEOFF_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
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_RETRANS MIT_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

Parameter Type	Data Type	Description/Values
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 logfile name to be opened for the given device or devices(s). 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.

bitmask
: bitmask to configure tracing for a given device or devices:
  - M3G_TRACE_H245 – decoded H.245 messages
  - M3G_TRACE_H223 – raw H.223 multiplexed bitstreams
  - M3G_TRACE_AUDIO – raw audio bitstreams
  - M3G_TRACE_VIDEO – raw video bitstreams
  - M3G_TRACE_INTERNALS – internal 3G-324M module debug tracing
  - M3G_TRACE_STATISTICS – 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

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_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.

Error Codes 5

This chapter describes the error codes used by the 3G-324M software.

The 3G-324M library functions return a value that indicates the success or failure of a function call. Success is indicated by M3G_SUCCESS, and failure is indicated by M3G_ERROR. If a function returns M3G_ERROR to indicate failure, use the Dialogic® Standard Runtime Library standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return the error description in a string format. These functions are described in the *Dialogic® Standard Runtime Library API Library Reference*.

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 3, "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 promotion of cellular systems that support high-speed data, known as third-generation (3G) systems. Established in 1998, 3GPP comprises North American, European, Japanese, Korean, and Chinese standards organizations.

3G-324M: Based on ITU-T H.324 recommendation modified by 3GPP for purposes of 3GPP circuit switched network based video telephony

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