Dialogic®

Dialogic® Multimedia API

Library Reference

March 2008

05-2454-001_D

Publication Date: March 2008

Document Number: 05-2454-001_D

Contents

Revision History 5

About This Publication 7

Purpose 7
Applicability 7
Intended Audience 7
How to Use This Publication 8
Related Information 8

1 Function Summary by Category 9

1.1 Multimedia API Header File 9
1.2 Dialogic® Multimedia API Device Management Functions 9
1.3 Configuration Functions 10
1.4 Input/Output Functions 10
1.5 Event Information Functions 11
1.6 Error Processing Functions 11

2 Function Information 13

2.1 Function Syntax Conventions 13
mm_Close( ) – close a previously opened multimedia device 14
mm_DisableEvents( ) – disable optional notification events 16
mm_EnableEvents( ) – enable optional notification events 18
mm_ErrorInfo( ) – retrieve current error information for a multimedia function 20
mm_GetMetaEvent( ) – get current SRL event information 22
mm_GetParm( ) – get the current configuration parameters 24
mm_Open( ) – open a multimedia device 26
mm_Play( ) – play a media object 29
mm_Record( ) – record a synchronized media object 33
mm_Reset( ) – reset an open multimedia device 38
mm_SetParm( ) – set the configuration parameters 41
mm_Stop( ) – stop the device operations 43

3 Events 47

3.1 Overview of Dialogic® Multimedia API Events 47
3.2 Dialogic® Multimedia API Event Types 48
3.3 Dialogic® Multimedia API Event Types by Function Type 49
3.4 Dialogic® Multimedia API Events 49

4 Data Structure Types 53

4.1 Overview of Dialogic® Multimedia API Data Structures 53
4.2 Data Structures for Function I/O 54
4.3 Data Structures for Analyzing Event Information 55
4.4 Play/Record Data Structure Levels 57
4.5 Other Data Structure Levels (_DETAILS) 57
4.6 Data Structures By Function 58

5 Data Structure Reference 61

MM_AUDIO_CODEC – audio codec specification 63
MM_ERROR_RESULT – error event information 64
MM_EVENTS – information for optional notification event functions 65
MM_GET_PARM – information for get parameter function 66
MM_GET_PARM_RESULT – retrieved parameter event information 67
MM_INFO – error or result information 68
MM_MEDIA_AUDIO – audio media item specification 69
MM_MEDIA_ITEM – media item specification 70
MM_MEDIA_ITEM_LIST – media item list information 71
MM_MEDIA_TERM – media termination information 72
MM_MEDIA_VIDEO – video media item specification 73
MM_METAEVENT – event descriptor for a metaevent 74
MM_PLAY_RECORD_CMPLT – play/record completion event information 76
MM_PLAY_RECORD_CMPLT_DETAILS – play/record completion details 77
MM_PLAY_RECORD_INFO – information for play and record functions 79
MM_PLAY_RECORD_LIST – list of items to play or record 80
MM_RET_CODE – error return code information 82
MM_SET_PARM – information for set parameter function 84
MM_STOP – information for stop device operations function 85
MM_STOP_ACK – stop ACK event information 87
MM_STOP_ACK_DETAILS – stop ACK detail information 88
MM_STOP_DETAILS – detailed stop request information 89
MM_VIDEO_CODEC – video codec specification 90
MM_VIDEO_RECORD_STARTED – I-Frame detection information 92

6 Multimedia API Errors 93

6.1 Overview of Dialogic® Multimedia API Errors 93
6.2 Dialogic® Multimedia API Function Error Codes 95
6.3 Dialogic® Multimedia API Event Information Error Return Codes 96
6.4 Terminating and Non-Terminating Play/Record Errors 97

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2454-001_D	March 2008	Made global changes to reflect Dialogic brand. mm_Play( ) function: Added note about supporting a single set of concurrent audio and video files. MM_MEDIA_AUDIO data structure: Added unOffset field. MM_VIDEO_CODEC data structure: Added note saying that none of the fields in the data structure may be modified for mm_Record( ).
05-2454-001	August 2005	Initial version of document published for Dialogic® Host Media Processing (HMP) Software Release 1.5 LIN.

About This Publication

The following topics provide information about this publication:

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

Purpose

This publication contains reference information for all functions, parameters, data structures, values, events, and error codes in the Multimedia API. The API provides the ability to record and play back digitized multimedia (audio and video) to support video services in application programs.

This publication is a companion document to the *Dialogic® Multimedia API Programming Guide*, which provides guidelines for developing applications using the Dialogic® Multimedia API.

Applicability

This document (05-2454-001_D) is published for Dialogic® Host Media Processing (HMP) Software Release 3.0WIN.

Intended Audience

This information is intended for:

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

How to Use This Publication

This publication assumes that you are familiar with and have prior experience with the operating system and the C programming language.

The information in this publication is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the categories of functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to all the functions in the library.
- Chapter 3, "Events" describes the events that are generated by the Multimedia API functions.
- Chapter 4, "Data Structure Types" describes the types of data structures supported by the Multimedia API, including the basic categories, specific types, naming conventions, purpose, hierarchy, and association with specific functions.
- Chapter 5, "Data Structure Reference" provides an alphabetical reference to the Multimedia API data structures, along with their fields and valid values.
- Chapter 6, "Multimedia API Errors" describes the errors that can be returned by the API.

Related Information

See the following for additional information:

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

Function Summary by Category 1

This chapter contains an overview of the Dialogic® Multimedia API functions and the categories into which they are logically grouped. Major topics include the following:

- Multimedia API Header File 9
- Dialogic® Multimedia API Device Management Functions 9
- Configuration Functions 10
- Input/Output Functions 10
- Event Information Functions 11
- Error Processing Functions 11

1.1 Multimedia API Header File

The Dialogic Multimedia API functions, parameters, data structures, values, events, and error codes are defined in the *mmlib.h* header file. The Multimedia API functions use an "mm_" prefix.

1.2 Dialogic® Multimedia API Device Management Functions

Dialogic® Device Management functions open and close channel devices.

Before you can call any other library function on a device, that device must be opened using a Device Management open function. The open function specifies the name of a device and returns a unique device handle, which contains a numerical reference to the device. This device handle is how the device is identified once the device has been opened. The handle is specified as an input parameter for any function that operates on that device. The close function is used to close a device and release its handle. Device Management functions do not cause a device to be busy and will work on a device whether the device is busy or idle.

The following naming convention is used for channel-level multimedia device names:

mmBnCy
: where "n" is the board device number assigned to the virtual multimedia board and "y" is the number of a multimedia channel device associated with that board. Examples of multimedia channel device names are mmB1C1 and mmB1C2.

See the *Dialogic® Standard Runtime Library API Programming Guide* for more information on device names, and see the *Dialogic® Multimedia API Programming Guide* for more information about using multimedia devices.

Note: This category of Device Management function is common among the APIs of the Dialogic® Software but is a completely different subject from the Device Management API library functions, which provide run-time control and management of configurable system devices.

mm_Close( )
Close a previously opened multimedia device

mm_Open( )
Open a multimedia device

mm_Reset( )
Reset an open multimedia device

1.3 Configuration Functions

Configuration functions allow you to alter, examine, and control the configuration of an open device. They operate on an idle device, cause the device to be busy, and return the device to an idle state once the configuration operation is complete.

Configuration functions can only be executed asynchronously. They return immediately to indicate successful initiation or an error. If successfully initiated, they perform their operations and stop, reporting either completion or failure through an appropriately named event. See the *Dialogic® Standard Runtime Library API Programming Guide* for more information on asynchronous operation.

mm_GetParm( )
Get the current configuration parameters

mm_SetParm( )
Set the configuration parameters

1.4 Input/Output Functions

The Input/Output functions control the transfer of data. Except for the **mm_Stop( )** function, the Input/Output functions transfer data to and from an open, idle channel; they cause a channel to be busy while data transfer is taking place and return the channel to an idle state when data transfer is complete.

Input/Output functions can only be executed asynchronously. They return immediately to indicate successful initiation or an error. If successfully initiated, they perform their operations, reporting any intermediate events, and then stop, reporting either completion or failure through an appropriately named event.

See the *Dialogic® Standard Runtime Library API Programming Guide* for more information on asynchronous operation.

mm_Play( )
Play a media object

mm_Record( )
Record a media object

mm_Stop( )
Stop the device operations

1.5 Event Information Functions

Event Information functions primarily provide information on Multimedia API events.

mm_DisableEvents( )
Disable optional notification events

mm_EnableEvents( )
Enable optional notification events

mm_GetMetaEvent( )
Get current SRL event information

1.6 Error Processing Functions

Error Processing functions provide Multimedia API error information.

mm_ErrorInfo( )
Retrieve current error information for a multimedia function

Function Information 2

This chapter is arranged in alphabetical order by function name and contains detailed information on each function in the Dialogic® Multimedia API library.

2.1 Function Syntax Conventions

The Dialogic Multimedia API library functions use the following format:

```
int mm_FunctionName (nDeviceHandle, Parameter1, Parameter2, ..., ParameterN)
```

where:

int
specifies integer as the return data type of the function.

mm_FunctionName
represents the function name. All Multimedia API functions use the “mm_” prefix.

nDeviceHandle
represents the device handle, which contains a numerical reference to a device. The device handle is obtained when the device is opened and must be specified as an input parameter for any function that operates on that device.

Parameter1, Parameter2, ..., ParameterN
represent input or output parameters

mm_Close( )

Name: int mm_Close(nDeviceHandle, pCloseInfo)

Inputs: int nDeviceHandle
- valid SRL handle representing a multimedia device

CPMM_CLOSE_INFO pCloseInfo
- pointer to close information structure. Must be set to NULL (**reserved for future use**).

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Device Management

Mode: Synchronous

Description

This function closes a multimedia device handle that was previously opened using **mm_Open( )**.

Note: This function does not affect any of the parameters that have been set for the device.

The **mm_Close( )** function stops any media operations on the device, but does not report these terminations with corresponding events (e.g., does not generate MMEV_PLAY or MMEV_PLAY_FAIL), because this function releases the handle on which the device events are reported. This function discards any outstanding events on the device handle and disables the generation of any new events on the handle.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pCloseInfo	points to MM_CLOSE_INFO structure. Must be set to NULL (**reserved for future use**).

Cautions

- The **pCloseInfo** pointer is **reserved for future use** and must be set to NULL.
- The only process affected by this function is the process that called the function. Once a device is closed, a process can no longer act on that device using that device handle.
- This function discards any outstanding events on that handle.
- This function disables the generation of all events on that handle.
- Do not use the operating system close command to close a multimedia device; unpredictable results will occur.

Errors

This function returns an EMM_ERROR if an incorrect parameter is specified.

If this function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. See Chapter 6, "Multimedia API Errors" for the function error codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   int nDeviceHandle; /* multimedia device handle */
   /* Main Processing
   .
   .
   */

   /*
    * Application is shutting down.
    * Need to close MM device handle.
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   if (mm_Close(nDeviceHandle, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_Open( )**
- **mm_Reset( )**
- **mm_Close( )**

mm_DisableEvents( )

Name: int mm_DisableEvents(nDeviceHandle, pEvents, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_EVENTS pEvents	• pointer to events information structure
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Event Information

Mode: Asynchronous

Description

This function disables optional notification events that are enabled by default or were previously enabled by **mm_EnableEvents( )**. It disables optional notification events only in the process in which it is called. Optional notification events are enabled by default and must specifically be disabled by the **mm_DisableEvents( )** function.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pEvents	points to MM_EVENTS structure
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_DISABLEEVENTS
Termination event reported upon successful completion of the function.
Event Data: MM_DISABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_DISABLEEVENTS_FAIL
Operation failure termination event reported upon encountering an error during the operation.
Event Data: MM_DISABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ERROR

Unsolicited event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

Cautions

None.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_EVENTS events;
   events.unVersion = MM_EVENTS_VERSION_0;
   events.unMask = MMR_EVENT_VIDEO_RECORD_STARTED;
   if (mm_DisableEvents(nDeviceHandle, &events, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_EnableEvents( )**
- **mm_GetMetaEvent( )**

mm_EnableEvents( )

Name: int mm_EnableEvents(nDeviceHandle, pEvents, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_EVENTS pEvents	• pointer to events information structure
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Event Information

Mode: Asynchronous

Description

This function enables optional notification events only in the process in which it is called. The optional notification events are enabled by default.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pEvents	points to MM_EVENTS structure
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_ENABLEEVENTS
Termination event reported upon successful completion of the function.
Event Data: MM_ENABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ENABLEEVENTS_FAIL
Operation failure termination event reported upon encountering an error during the operation.
Event Data: MM_ENABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ERROR
Unsolicited event reported upon encountering an unexpected failure.
Event Data: MM_ERROR_RESULT structure

Cautions

None.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_EVENTS events;
   events.unVersion = MM_EVENTS_VERSION_0;
   events.unMask = MMR_EVENT_VIDEO_RECORD_STARTED;
   if (mm_EnableEvents(nDeviceHandle, &events, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_DisableEvents( )**
- **mm_GetMetaEvent( )**

mm_ErrorInfo( )

Name: int mm_ErrorInfo(pInfo)

Inputs: PMM_INFO pInfo • pointer to the MM_INFO data structure

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Error Processing

Mode: Synchronous

Description

This function obtains the error information for the last error that occurred in the Dialogic® Multimedia API and provides it in the MM_INFO structure. To retrieve the error information about a failed function, the **mm_ErrorInfo( )** function must be called immediately after the Dialogic Multimedia API function failed.

Parameter	Description
pInfo	points to the MM_INFO structure where information about the error is contained.

Cautions

- The **mm_ErrorInfo( )** function can only be called in the same thread in which the routine that had the error was called. The **mm_ErrorInfo( )** function cannot be called to retrieve error information for a function that returned error information in another thread.
- Because the Multimedia API keeps the error information for the last Multimedia API function call that sets it, the error information may be changed by succeeding Multimedia API calls. Therefore it is recommended to check and retrieve error information immediately after a Multimedia API function fails.

Errors

This function returns an EMM_ERROR if an incorrect parameter is specified. The **mm_ErrorInfo( )** function should not be called recursively if it returns EMM_ERROR to indicate failure. An EMM_ERROR generally indicates that **pInfo** is NULL or invalid.

Example

```
#include <mmlib.h>

/*
 * This function is called to print MM_INFO to the system console
 * Typically it would be called after a call to mm_ErrorInfo
 * to print the resulting MM_INFO data structure
 */
void PrintMM_INFO(const MM_INFO* pInfo)
```

```
{
   printf("mmValue = %#x\n", pInfo->mmValue);
   printf("mmMsg = %s\n", pInfo->mmMsg);
   printf("additionalInfo = %s\n", pInfo->additionalInfo);
}

/*
 * This function can be called anytime an error occurs
 * Not shown is an MM function which fails and calls this function
 * This procedure prints error information to the console with no other side effects
 */
void PrintErrorInfo(void)
{
   int retCode;
   MM_INFO t_Info;
   t_Info.unVersion = MM_INFO_VERSION_0;

   retCode = mm_ErrorInfo(&t_Info);
   if (retCode == EMM_SUCCESS)
   {
      printf("mm_ErrorInfo successfully called\n");
      PrintMM_INFO(&t_Info);
   }
   else
   {
      printf("mm_ErrorInfo call failed\n");
   }
}

int main(int argc, char* argv[])
{
   .
   .
   /*
    * Calling an MM function with error processing
    */
   if (mm_DisableEvents(nDeviceHandle, &events, NULL) == EMM_ERROR)
   {
      /* process error */
      PrintErrorInfo();
   }
}
```

See Also

- **mm_GetMetaEvent( )**
- Chapter 6, "Multimedia API Errors"

mm_GetMetaEvent( )

Name: int mm_GetMetaEvent(pMetaEvent)

Inputs: MM_METAEVENT pMetaEvent
- pointer to MM_METAEVENT data structure of metaevent data

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Event Information

Mode: Synchronous

Description

This function retrieves event information for the current Dialogic® Standard Runtime Library (SRL) event that stores the Dialogic® Multimedia API and non-Multimedia API event information. The MM_METAEVENT data structure contains explicit information describing the SRL event to be returned to the application program. This data structure provides uniform information retrieval among call control libraries and across operating systems.

You must call the **mm_GetMetaEvent( )** function to retrieve any Dialogic Multimedia API event information and any other event information if you are not sure of the event type. If the metaevent is a Multimedia API event, the MMME_MM_EVENT bit in the MM_METAEVENT flags field will be set. The MM_METAEVENT fields contain valid Multimedia API-related data only when the MMME_MM_EVENT bit is set. Do **not** use these fields for obtaining multimedia information if the bit is not set.

The current SRL event information is not changed or altered by calling the **mm_GetMetaEvent( )** function to retrieve event information. This function may be used as a convenience function to retrieve the event information for all SRL events. Whether the event is a Multimedia API event or any other SRL event, the SRL event information (for example, evtdatap, evttype) may be retrieved from the MM_METAEVENT data structure instead of using SRL functions to retrieve this information. For information on the SRL API, see the *Dialogic® Standard Runtime Library API Library Reference*.

Parameter	Description
pMetaEvent	points to the MM_METAEVENT structure filled by this function

Cautions

- The **mm_GetMetaEvent( )** function **must** be the first function called before processing any Multimedia API event.
- An application should call the **mm_GetMetaEvent( )** function only once for a given event. Calling the function more than once will result in data corruption or an access violation.

- The event must be processed entirely in the same thread or all information about the event must be retrieved before processing the event in another thread.

Errors

This function returns an EMM_ERROR if an incorrect parameter is specified.

If this function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. See Chapter 6, "Multimedia API Errors" for the function error codes.

Example

```
MM_METAEVENT metaevent;
metaevent.unVersion = MM_METAEVENT_VERSION_0;
if (sr_waitevt(timeout) != -1)
{
   if (mm_GetMetaEvent(&metaevent) == EMM_ERROR)
   {
      /* get and process the error */
   }
   else
   {
      /* Process retrieved metaevent */
   }
}
```

See Also

-
- **mm_DisableEvents( )**
- **mm_EnableEvents( )**

mm_GetParm( )

Name: int mm_GetParm(nDeviceHandle, pGetParm, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_GET_PARM pGetParm	• pointer to get parameters information structure
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Configuration

Mode: Asynchronous

Description

This function gets the value of the specified configuration parameters.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pGetParm	points to MM_GET_PARM information structure
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_GETPARM
Termination event reported upon successful completion of the function.
Event Data: MM_GET_PARM_RESULT structure

MMEV_GETPARM_FAIL
Operation failure termination event reported upon encountering an error during the operation.
Event Data: MM_GET_PARM_RESULT structure

MMEV_ERROR
Unsolicited event reported upon encountering an unexpected failure.
Event Data: MM_ERROR_RESULT structure

Cautions

- If an MMEV_GET_PARM_FAIL event is generated, the MM_GET_PARM_RESULT data structure contains invalid data.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_GET_PARM parm;
   parm.unVersion = MM_GET_PARM_VERSION_0;

   parm.eParm = MM_PARM_REC_IFRAME_TIMEOUT;
   if (mm_GetParm(nDeviceHandle, &parm, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_SetParm( )**

mm_Open( )

Name: int mm_Open(szDevName, pOpenInfo, pUserInfo)

Inputs:

const char* szDevName	• pointer to device name to open
CPMM_OPEN_INFO pOpenInfo	• pointer to open information structure. Must be set to NULL (**reserved for future use**).
void* pUserInfo	• pointer to user-defined buffer

Returns: **device handle** if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Device Management

Mode: Asynchronous

Description

This function opens a multimedia device and returns a unique Standard Runtime Library (SRL) device handle to identify the device. All subsequent references to the opened device must be made using the handle until the device is closed. For information on the SRL API, see the *Dialogic® Standard Runtime Library API Library Reference*.

The device handle returned by this function is defined by Dialogic. It is not a standard operating system file descriptor.

Parameter	Description
szDevName	points to a multimedia device name to open. The following naming convention is used for channel-level multimedia device names: • mmBnCy – where "n" is the board device number assigned to the virtual multimedia board and "y" is the number of a multimedia channel device associated with that board. Examples of multimedia channel device names are mmB1C1 and mmB1C2.
pOpenInfo	points to MM_OPEN_INFO structure. Must be set to NULL (**reserved for future use**).
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

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

If this function is called and it generates an MMEV_OPEN_FAIL event, a device handle is returned, but the application must close the handle by calling **mm_Close( )**.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, "Events".

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_OPEN
: Termination event reported upon successful completion of the function.

Event Data: MM_OPEN_RESULT structure (MM_RET_CODE)

MMEV_OPEN_FAIL
: Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_OPEN_RESULT structure (MM_RET_CODE)

Note: The application program must call **mm_Close( )** to clean up after this failure.

MMEV_ERROR
: Unsolicited event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

Cautions

- You must obtain an MMEV_OPEN event before using the device handle.
- You must close the handle by calling **mm_Close( )** if the function generates an MMEV_OPEN_FAIL event.
- Do not use the operating system open function to open multimedia devices; unpredictable results will occur.
- The **pOpenInfo** pointer is **reserved for future use** and must be set to NULL.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   int nDeviceHandle; /* multimedia device handle */
   /* Open Device */
   if ((nDeviceHandle = mm_Open("mmB1C1", NULL, NULL)) == EMM_ERROR) {
      /* process error */
}
.
.
.
```

See Also

- **mm_Close( )**
- **mm_Reset( )**

mm_Play( )

Name: int mm_Play(nDeviceHandle, pPlayInfo, pRuntimeControl, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_PLAY_INFO pPlayInfo	• pointer to play information structure
CPMM_RUNTIME_CONTROL pRuntimeControl	• pointer to runtime control information structure. Must be set to NULL (**reserved for future use**).
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: I/O

Mode: Asynchronous

Description

This function plays back audio and video data from multimedia files while maintaining their synchronization. It can also play back only the audio portion or video portion.

The multimedia files consist of a Dialogic® Multimedia API audio file and video file as specified in the MM_MEDIA_AUDIO and MM_MEDIA_VIDEO structures. When performing multimedia recording or playback, the video data is synchronized with the audio data.

Note: This function currently supports a single set of concurrent audio and video files. It does not support multiple sets of audio and video files.

This function can transmit a tone or "beep" to indicate the start of playback. See MM_MEDIA_AUDIO and MM_MEDIA_VIDEO for more information.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pPlayInfo	points to MM_PLAY_INFO (MM_PLAY_RECORD_INFO) structure
pRuntimeControl	points to MM_RUNTIME_CONTROL structure. Must be set to NULL (**reserved for future use**).
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )**

function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, "Events".

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_PLAY_ACK
Initiation event reported upon successful start of the function.
Event Data: MM_PLAY_ACK structure (MM_RET_CODE)

MMEV_PLAY_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.
Event Data: MM_PLAY_ACK structure (MM_RET_CODE)

MMEV_PLAY
Termination event reported upon successful completion or successful termination of the operation.
Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)
After a play operation has begun, an MMEV_PLAY event is reported to the application program when one of the following conditions occurs:

- the device finishes playing the media specified in the **pPlayInfo** parameter; i.e., it reaches the end of file
- the application program issues a stop operation with **mm_Stop( )**. See the **mm_Stop( )** function for conditions that generate no completion events or multiple completion events.
- the application program issues a reset operation with **mm_Reset( )**

MMEV_PLAY_FAIL
Operation failure termination event reported upon encountering an error during the operation.
Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)
Note: See Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for related information.

MMEV_ERROR
Unsolicited event reported upon encountering an unexpected failure.
Event Data: MM_ERROR_RESULT structure

Note: For more information on these events, see Chapter 3, "Events". See also Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for useful information.

Cautions

- See the MMEV_PLAY event above for conditions under which you can receive more than one MMEV_PLAY event.
- The **pRuntimeControl** pointer is **reserved for future use** and must be set to NULL.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_PLAY_INFO play_info;
   play_info.unVersion = MM_PLAY_RECORD_INFO_VERSION_0;
   MM_PLAY_RECORD_LIST playlist[2];

   MM_MEDIA_ITEM_LIST mediaitemlist1[1];
   MM_MEDIA_ITEM_LIST mediaitemlist2[1];
   const MM_VIDEO_CODEC VideoCodecType1 = {
      MM_VIDEO_CODEC_VERSION_0,
      EMM_VIDEO_CODING_DEFAULT,
      EMM_VIDEO_PROFILE_DEFAULT,
      EMM_VIDEO_LEVEL_DEFAULT,
      EMM_VIDEO_IMAGE_WIDTH_DEFAULT,
      EMM_VIDEO_IMAGE_HEIGHT_DEFAULT,
      EMM_VIDEO_BITRATE_DEFAULT,
      EMM_VIDEO_FRAMESPERSEC_DEFAULT
   };
   const MM_AUDIO_CODEC AudioCodecType1 = {
      MM_AUDIO_CODEC_VERSION_0,
      MM_DATA_FORMAT_PCM,
      MM_DRT_8KHZ,
      16
   };
   const char VideoFileName1[] = "/dir/file1.vid";
   const char AudioFileName1[] = "/dir/file3.aud";

   int cc;
   int xx;

   cc = 0;
   // Build Video Item 1
   mediaitemlist1[cc].unVersion = MM_MEDIA_ITEM_LIST_VERSION_0;
   mediaitemlist1[cc].ItemChain = EMM_ITEM_EOT;
   mediaitemlist1[cc].item.video.codec = VideoCodecType1;
   mediaitemlist1[cc].item.video.unMode = 0;
   mediaitemlist1[cc].item.video.szFileName = VideoFileName1;
   cc++;

   xx = 0;
   // Add Video Items to the PlayList
   playlist[xx].unVersion = MM_PLAY_RECORD_LIST_VERSION_0;
   playlist[xx].ItemChain = EMM_ITEM_CONT;
   playlist[xx].ItemType = EMM_MEDIA_TYPE_VIDEO;
   playlist[xx].list = mediaitemlist1;
   xx++;

   cc = 0;
   // Build Audio Item 1
   mediaitemlist2[cc].unVersion = MM_MEDIA_ITEM_LIST_VERSION_0;
   mediaitemlist2[cc].ItemChain = EMM_ITEM_EOT;
   mediaitemlist2[cc].item.audio.codec = AudioCodecType1;
   mediaitemlist2[cc].item.audio.unMode = MM_MODE_AUD_FILE_TYPE_VOX;
   mediaitemlist2[cc].item.audio.ulOffset = 0;
   mediaitemlist2[cc].item.audio.szFileName = AudioFileName1;
   cc++;
```

```
    // Add Audio Items to the PlayList
    playlist[xx].unVersion = MM_PLAY_RECORD_LIST_VERSION_0;
    playlist[xx].ItemChain = EMM_ITEM_EOT;
    playlist[xx].ItemType = EMM_MEDIA_TYPE_AUDIO;
    playlist[xx].list = mediaitemlist2;
    xx++;

    // Form Play Info
    play_info.eFileFormat = EMM_FILE_FORMAT_PROPRIETARY;
    play_info.list = playlist;

    // Initiate Play
    If (mm_Play(nDeviceHandle, &play_info, NULL, NULL) == EMM_ERROR)
    {
       /* process error */
    }
}
```

See Also

- **mm_Record( )**
- **mm_Reset( )**
- **mm_Stop( )**

mm_Record( )

Name: int mm_Record(nDeviceHandle, pRecordInfo, pRuntimeControl, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_RECORD_INFO pRecordInfo	• pointer to record information structure
CPMM_RUNTIME_CONTROL pRuntimeControl	• pointer to runtime control information structure. Must be set to NULL (**reserved for future use**).
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: I/O

Mode: Asynchronous

Description

This function records synchronized audio and video data to multimedia files. It can also record only the audio portion or video portion

The multimedia files consist of a Dialogic® Multimedia API audio file and video file as specified in the MM_MEDIA_AUDIO and MM_MEDIA_VIDEO structures. When performing multimedia recording or playback, the video data is synchronized with the audio data.

This function can transmit a start-of-recording tone or "beep" to notify the party being recorded. See MM_MEDIA_AUDIO and MM_MEDIA_VIDEO for more information.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pRecordInfo	points to MM_RECORD_INFO (MM_PLAY_RECORD_INFO) structure
pRuntimeControl	points to MM_RUNTIME_CONTROL structure. Must be set to NULL (**reserved for future use**).
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Detection of an I-frame (complete video frame) can be used to trigger recording. The API controls I-frame detection and its responses through the following:

Video Record Beep

On I-frame detection, transmit a beep to the party being recorded (start-of-recording notification tone). The video record beep is enabled by default and is transmitted upon detection of an I-frame or upon time-out waiting for an I-frame. The beep can be disabled by specifying the MM_MODE_VID_NOIFRMBEEPINITIATED bit in the MM_MEDIA_VIDEO unMode field.

I-Frame Time-Out Parameter

The EMM_REC_IFRAME_TIMEOUT parameter controls the time to wait for an I-frame. Video recording, or multimedia (audio and video) recording, starts when an I-frame is detected or when the time-out is reached. The default time-out is 5000 ms (5 seconds). To change this, specify the EMM_REC_IFRAME_TIMEOUT parameter in the MM_SET_PARM eParm field, specify a value in the unParmValue field, and call **mm_SetParm( )** to set it. A setting of 0 (zero) causes an immediate time-out and starts recording immediately.

To get the current setting of the I-frame time-out parameter, specify EMM_REC_IFRAME_TIMEOUT in the MM_GET_PARM eParm field and call **mm_GetParm( )**. When the function generates an MMEV_GETPARM event, the associated event data provides the current setting in the MM_GET_PARM_RESULT unParmValue field.

MMEV_VIDEO_RECORD_STARTED event

This ***optional*** **intermediate (non-terminating) notification event** is enabled by default and indicates the actual start of video recording (I-frame received or time-out waiting for an I-frame). This event can be disabled by **mm_DisableEvents( )** and enabled by **mm_EnableEvents( )** by specifying the MMR_EVENT_VIDEO_RECORD_STARTED bit in the MM_EVENTS unMask field.

The event data associated with the event indicates the status in the MM_VIDEO_RECORD_STARTED unStatus field. The field indicates EMM_VIDEO_RCRD_IFRAME_DETECTED for recording started due to I-frame detection, and EMM_VIDEO_RCRD_IFRAME_TIMEOUT for recording started due to I-frame time-out.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_RECORD_ACK

Initiation event reported upon successful start of the function.

Event Data: MM_RECORD_ACK structure (MM_RET_CODE)

MMEV_RECORD_ACK_FAIL

Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_RECORD_ACK structure (MM_RET_CODE)

MMEV_VIDEO_RECORD_STARTED

Optional **intermediate (non-terminating) notification event** (enabled by default), indicating actual start of recording (complete video frame, or I-frame, received, or time-out waiting for an I-frame). This event can be disabled by **mm_DisableEvents( )** and enabled by **mm_EnableEvents( )**.

Event Data: MM_VIDEO_RECORD_STARTED structure

MMEV_VIDEO_RECORD_STARTED_FAIL (**reserved for future use**)
***Optional* intermediate (non-terminating) failure notification event** that is **reserved for future use** (there are no conditions that generate it).

MMEV_RECORD
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

After a record operation has begun, an MMEV_RECORD event is reported to the application program when one of the following conditions occurs:

- the application program issues a stop operation with **mm_Stop( )**. See the **mm_Stop( )** function for conditions that generate no completion events or multiple completion events.
- the application program issues a reset operation with **mm_Reset( )**

MMEV_RECORD_FAIL
Operation failure termination event indicating operation errors reported upon completion or termination of the operation. The operation errors that can cause this failure event are **intermediate (non-terminating) errors**.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_ERROR
Unsolicited event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

Note: For more information on these events, see Chapter 3, "Events". See also Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for useful information.

Cautions

- See the MMEV_RECORD event above for conditions under which you can receive more than one MMEV_RECORD event.
- The **pRuntimeControl** pointer is **reserved for future use** and must be set to NULL.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_RECORD_INFO record_info;
   record_info.unVersion = MM_PLAY_RECORD_INFO_VERSION_0;
   MM_PLAY_RECORD_LIST recordlist[2];
```

```
    MM_MEDIA_ITEM_LIST mediaitemlist1[1];
    MM_MEDIA_ITEM_LIST mediaitemlist2[1];
    const MM_VIDEO_CODEC VideoCodecType1 = {
       MM_VIDEO_CODEC_VERSION_0,
       EMM_VIDEO_CODING_DEFAULT,
       EMM_VIDEO_PROFILE_DEFAULT,
       EMM_VIDEO_LEVEL_DEFAULT,
       EMM_VIDEO_IMAGE_WIDTH_DEFAULT,
       EMM_VIDEO_IMAGE_HEIGHT_DEFAULT,
       EMM_VIDEO_BITRATE_DEFAULT,
       EMM_VIDEO_FRAMESPERSEC_DEFAULT
    };
    const MM_AUDIO_CODEC AudioCodecType1 = {
       MM_DATA_FORMAT_PCM,
       MM_DRT_8KHZ,
       16
    };
    const char VideoFileName1[] = "/dir/file1.vid";
    const char AudioFileName1[] = "/dir/file3.aud";

    int cc;
    int xx;

    cc = 0;
    // Build Video Item 1
    mediaitemlist1[cc].unVersion = MM_MEDIA_ITEM_LIST_VERSION_0;
    mediaitemlist1[cc].ItemChain = EMM_ITEM_EOT;
    mediaitemlist1[cc].item.video.codec = VideoCodecType1;
    mediaitemlist1[cc].item.video.unMode = 0;
    mediaitemlist1[cc].item.video.szFileName = VideoFileName1;
    cc++;

    xx = 0;
    // Add Video Items to the RecordList
    recordlist[xx].unVersion = MM_PLAY_RECORD_LIST_VERSION_0;
    recordlist[xx].ItemChain = EMM_ITEM_CONT;
    recordlist[xx].ItemType = EMM_MEDIA_TYPE_VIDEO;
    recordlist[xx].list = mediaitemlist1;
    xx++;

    cc = 0;
    // Build Audio Item 1
    mediaitemlist2[cc].unVersion = MM_MEDIA_ITEM_LIST_VERSION_0;
    mediaitemlist2[cc].ItemChain = EMM_ITEM_EOT;
    mediaitemlist2[cc].item.audio.codec = AudioCodecType1;
    mediaitemlist2[cc].item.audio.unMode = MM_MODE_AUD_FILE_TYPE_VOX;
    mediaitemlist2[cc].item.audio.ulOffset = 0;
    mediaitemlist2[cc].item.audio.szFileName = AudioFileName1;
    cc++;

    // Add Audio Items to the PlayList
    recordlist[xx].unVersion = MM_PLAY_RECORD_LIST_VERSION_0;
    recordlist[xx].ItemChain = EMM_ITEM_EOT;
    recordlist[xx].ItemType = EMM_MEDIA_TYPE_AUDIO;
    recordlist[xx].list = mediaitemlist2;
    xx++;

    // Form Record Info
    record_info.eFileFormat = EMM_FILE_FORMAT_PROPRIETARY;
    record_info.list = recordlist;

    // Initiate Record
    if (mm_Record(nDeviceHandle, &record_info, NULL, NULL) == MM_ERROR)
    {
       /* process error */
    }
}
```

See Also

- **mm_Play( )**
- **mm_Reset( )**
- **mm_Stop( )**

mm_Reset( )

Name: int mm_Reset(nDeviceHandle, pReset, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_RESET pReset	• pointer to reset information structure. Must be set to NULL (**reserved for future use**).
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: I/O

Mode: Asynchronous

Description

This function terminates all active media on the device and resets the device state to idle. All previously set parameters are removed and the defaults are applied. The state of the device is equivalent to the state after the device was first opened, except that **mm_Reset( )** does not reset any device connections made through the Dialogic® Device Management API.

If this function stops an operation that is in progress (i.e., busy), it generates a termination event corresponding to the operation (e.g., MMEV_PLAY or MMEV_PLAY_FAIL) in addition to any reset-specific events (MMEV_RESET_ACK, MMEV_RESET_ACK_FAIL, MMEV_RESET, MMEV_RESET_FAIL). See Asynchronous Mode Events below.

If user information is passed in the **mm_Reset( ) pUserInfo** parameter, it only gets passed to the metaevent data for the reset-specific events and not the media operation events, which can have their own user information associated with them.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pReset	points to MM_RESET structure. Must be set to NULL (**reserved for future use**).
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_RESET_ACK
Initiation event reported upon successful start of the function.

Event Data: MM_RESET_ACK structure (MM_RET_CODE)

MMEV_RESET_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_RESET_ACK structure (MM_RET_CODE)

MMEV_RESET
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_RESET_RESULT structure (MM_RET_CODE)

MMEV_RESET_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_RESET_RESULT structure (MM_RET_CODE)

MMEV_PLAY
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_PLAY_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

Note: See Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for related information.

MMEV_RECORD
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_RECORD_FAIL
Operation failure termination event indicating operation errors reported upon completion or termination of the operation. The operation errors that can cause this failure event are **intermediate (non-terminating) errors**.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_ERROR
Unsolicited event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

Cautions

- The **pReset** pointer is **reserved for future use** and must be set to NULL.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   int nDeviceHandle; /* multimedia device handle */
   /*
   .
   .
   Main Processing
   .
   .
   */

   /*
    * Application is shutting down.
    * Need to close MM device handle.
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   if (mm_Reset(nDeviceHandle, NULL, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_Open( )**
- **mm_Close( )**

mm_SetParm( )

Name: int mm_SetParm(nDeviceHandle, pSetParm, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_SET_PARM pSetParm	• pointer to set parameters information structure
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: Configuration

Mode: Asynchronous

Description

This function sets the value of the specified configuration parameters. Parameters that are set remain in effect even after an **mm_Close( )**. However, an **mm_Reset( )** will reset all parameters to their default values.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pSetParm	points to MM_SET_PARM structure
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_SETPARM
Termination event reported upon successful completion of the function.
Event Data: MM_SET_PARM_RESULT structure (MM_RET_CODE)

MMEV_SETPARM_FAIL
Operation failure termination event reported upon encountering an error during the operation.
Event Data: MM_SET_PARM_RESULT structure (MM_RET_CODE)

MMEV_ERROR

Unsolicited event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

Cautions

None.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_SET_PARM parm;
   parm.unVersion = MM_SET_PARM_VERSION_0;

   parm.eParm = MM_PARM_REC_IFRAME_TIMEOUT;
   parm.unParmValue = 2222;
   if (mm_SetParm(nDeviceHandle, &parm, NULL) == EMM_ERROR)
   {
      /* process error */
   }
}
```

See Also

- **mm_GetParm( )**
- **mm_Reset( )**

mm_Stop( )

Name: int mm_Stop(nDeviceHandle, pStop, pUserInfo)

Inputs:

int nDeviceHandle	• valid SRL handle representing a multimedia device
CPMM_STOP pStop	• pointer to stop information structure
void* pUserInfo	• pointer to user-defined buffer

Returns: EMM_SUCCESS if successful
EMM_ERROR if failure

Includes: mmlib.h

Category: I/O

Mode: Asynchronous

Description

This function forces termination of specified multimedia I/O functions or operations on a channel. For example, this function can stop selected **mm_Play( )** or **mm_Record( )** operations (items) such as a video play, video record, audio play, and audio record. The operation or item type is specified in the MM_STOP structure. To stop all types of multimedia I/O operations on the channel, you must specify each item.

Parameter	Description
nDeviceHandle	specifies a valid multimedia device handle obtained from a previous open
pStop	points to MM_STOP structure
pUserInfo	points to user-defined buffer. See MM_METAEVENT evtUserInfo field.

If a specified operation (item) is not currently active on the channel, the function completes successfully for that operation but has no effect. If all possible operations are successfully stopped on a channel that is busy, it forces the channel to become idle. If the channel is already idle, it completes successfully but has no effect.

Regardless of whether the function stops an operation that is idle or in progress (i.e., busy), it generates an MM_STOP_ACK event to indicate successful initiation. If there are no busy operations, *this is the only event generated*. However, if a busy operation is stopped, it generates a corresponding event (e.g., MMEV_PLAY or MMEV_PLAY_FAIL) in addition to any stop-specific events (MMEV_STOP_ACK or MMEV_STOP_ACK_FAIL). See Asynchronous Mode Events below.

If the application program stops a selected media type (e.g., ItemType EMM_STOP_VIDEO_PLAY), it will receive an MMEV_PLAY event upon completion of the stop and the rest of the media types (if any; e.g., audio play) will continue to play until termination or completion of the play, at which time, it will generate another MMEV_PLAY event. Therefore, *it is possible to receive multiple MMEV_PLAY events* resulting from one **mm_Play( )** function.

Similarly, if the program stops a selected recording media type (e.g., ItemType EMM_STOP_VIDEO_RECORD), it will receive an MMEV_RECORD event upon completion of the stop, and the rest of the media types (if any; e.g., audio record) will continue to record until termination or completion, at which time, it will generate another MMEV_RECORD event. Therefore, *it is possible to receive multiple MMEV_RECORD events* resulting from one **mm_Record( )** function.

If an error is encountered for any of the selected stop items, none of the operations will be stopped (no matter where in the list the error occurred). To identify which item caused the error, use the MM_STOP_ACK and MM_STOP_ACK_DETAILS event information structures associated with the MMEV_STOP_ACK_FAIL event. The MM_STOP_ACK_DETAILS structure gives the item type and the error return code for each item specified in MM_STOP. An EMMRC_OK error return code indicates that no error occurred for that particular operation.

If user information is passed in the **mm_Stop( ) pUserInfo** parameter, it only gets passed to the metaevent data for the stop-specific events. It does not get passed to the media operation events, which can have their own user information associated with them.

Asynchronous Mode Events

Use the Dialogic® Standard Runtime Library (SRL) functions to process the events. Use the Multimedia API **mm_GetMetaEvent( )** Event Information function to retrieve the event information. Event data is indicated for the following events where applicable and can be obtained through the MM_METAEVENT structure eventdatap field after calling the **mm_GetMetaEvent( )** function. The event data is valid only until the next **mm_GetMetaEvent( )** is called. For more information on these events, see Chapter 3, “Events”.

If the function returns EMM_SUCCESS, it can generate any of the following events:

MMEV_STOP_ACK
: Initiation event reported upon successful start of the function.

  Event Data: MM_STOP_ACK structure

MMEV_STOP_ACK_FAIL
: Initiation failure termination event reported upon encountering an error before the operation begins.

  Event Data: MM_STOP_ACK structure

MMEV_PLAY
: Termination event reported upon successful completion or successful termination of the operation.

  Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_PLAY_FAIL
: Operation failure termination event reported upon encountering an error during the operation.

  Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

  Note: See Section 6.4, “Terminating and Non-Terminating Play/Record Errors”, on page 97 for related information.

MMEV_RECORD
: Termination event reported upon successful completion or successful termination of the operation.

  Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_RECORD_FAIL
: Operation failure termination event indicating operation errors reported upon completion or termination of the operation. The operation errors that can cause this failure event are **intermediate (non-terminating) errors**.

  Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_ERROR
: Unsolicited event reported upon encountering an unexpected failure.

  Event Data: MM_ERROR_RESULT structure

Cautions

- If there are no busy operations, the MM_STOP_ACK event (indicating successful initiation) is the only event generated.

Errors

If a Dialogic® Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. If the function generates a failure event, use the **mm_GetMetaEvent( )** function to obtain the error information. See Chapter 6, "Multimedia API Errors" for the function error codes and event information error return codes.

Example

```
#include <mmlib.h>
int main(int argc, char* argv[])
{
   .
   .
   /*
    * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to mm_Open( ).
    */
   MM_STOP stop_info[4];

   stop_info[0].unVersion = MM_STOP_VERSION_0;
   stop_info[0].ItemChain = EMM_ITEM_CONT;
   stop_info[0].ItemType = EMM_STOP_VIDEO_PLAY;

   stop_info[1].unVersion = MM_STOP_VERSION_0;
   stop_info[1].ItemChain = EMM_ITEM_CONT;
   stop_info[1].ItemType = EMM_STOP_VIDEO_RECORD;

   stop_info[2].unVersion = MM_STOP_VERSION_0;
   stop_info[2].ItemChain = EMM_ITEM_CONT;
   stop_info[2].ItemType = EMM_STOP_AUDIO_PLAY;

   stop_info[3].unVersion = MM_STOP_VERSION_0;
   stop_info[3].ItemChain = EMM_ITEM_CONT;
   stop_info[3].ItemType = EMM_STOP_AUDIO_RECORD;
```

```
    if (mm_Stop(nDeviceHandle, stop_info, NULL) == EMM_ERROR)
    {
       /* process error */
    }
}
```

See Also

- **mm_Play( )**
- **mm_Record( )**
- **mm_Reset( )**

Events 3

This chapter provides information about the events that are generated by the Dialogic® Multimedia API library functions.

- Overview of Dialogic® Multimedia API Events. 47
- Dialogic® Multimedia API Event Types. 48
- Dialogic® Multimedia API Event Types by Function Type 49
- Dialogic® Multimedia API Events 49

3.1 Overview of Dialogic® Multimedia API Events

An event indicates that a specific activity has occurred on a channel. The API reports channel activity to the application program in the form of events, which allows the program to identify and respond to a specific occurrence on a channel. Events provide feedback on the progress and completion of functions and indicate the occurrence of other channel activities. Events are sometimes referred to according to the type of event, such as initiation event, intermediate event, notification event, termination or completion event, unsolicited event, success event, and failure event. These types are not always mutually exclusive. The most common type of event is one that reports on the result of function operations. Typically, each function generates different events, and the functions documented in Chapter 2, "Function Information" describe the events applicable to them.

To collect an event code, use **sr_waitevt( )**, **sr_enbhdlr( )** or other Dialogic® Standard Runtime Library (SRL) function, depending upon the programming model in use. For detailed information on event handling and management, see the *Dialogic® Standard Runtime Library API Library Reference* and the *Dialogic® Standard Runtime Library API Programming Guide*.

The Multimedia API Event Information functions are listed in Section 1.5, "Event Information Functions", on page 11. The **mm_GetMetaEvent( )** function maps the current SRL event into an MM_METAEVENT data structure, which contains explicit data describing the event. This data mechanism helps to provide uniform information retrieval among libraries.

For Dialogic Multimedia API events, see the **mm_GetMetaEvent( )** function and the MM_METAEVENT structure for the specific multimedia information provided. This mechanism can also be used for non-Multimedia API events, for which the MM_METAEVENT structure provides the device descriptor, the event type, a pointer to variable length event data, and the length of the event data. No additional SRL calls are required to access event data, because all the data associated with any type of event are accessible via the MM_METAEVENT structure.

The **mm_EnableEvents( )** and **mm_DisableEvents( )** functions allow you to enable and disable optional notification events, such as the MMEV_VIDEO_RECORD_STARTED event, which indicates the actual start of recording.

Note: The MMEV_VIDEO_RECORD_STARTED event is enabled by default.

3.2 Dialogic® Multimedia API Event Types

The API can generate the following types of events:

Initiation Events

Format: **MMEV_xxxx_ACK** (where "xxxx" is the name of the function or operation)

These intermediate events are generated upon successful start of a function. For example, MMEV_PLAY_ACK indicates the successful start of the **mm_Play( )** function. The event data provides information about the reason for the event.

Initiation Failure Termination Events

Format: **MMEV_xxxx_ACK_FAIL** (where "xxxx" is the name of the function or operation)

These termination events are generated upon encountering an error before the main operation of a function begins. For example, MMEV_PLAY_ACK_FAIL indicates that a failure occurred before starting the **mm_Play( )** playback operation. The event data provides error information about the reason for the failure.

Completion or Successful Termination Events

Format: **MMEV_xxxx** (where "xxxx" is the name of the function or operation)

These termination events are generated upon the successful completion or successful termination of a function operation. For example, MMEV_PLAY indicates successful completion of the playback operation being performed by **mm_Play( )**; e.g., the end of file was reached. The deliberate termination of an I/O function operation by **mm_Stop( )** or **mm_Reset( )** can also produce this type of event; for example, MMEV_PLAY can indicate the successful termination of an **mm_Play( )** playback operation by **mm_Stop( )**. The event data provides result information about the reason for the completion or termination event.

Operation Failure Termination Events

Format: **MMEV_xxxx_FAIL** (where "xxxx" is the name of the function or operation)

These termination events are generated upon encountering an error during the main operation of a function. (Or, in the in the case of MMEV_RECORD_FAIL, upon completion or termination of the operation.) For example, MMEV_PLAY_FAIL indicates failure of the playback operation being performed by **mm_Play( )**. The deliberate termination of an I/O function operation by **mm_Stop( )** or **mm_Reset( )** can also produce this type of event; for example, MMEV_PLAY_FAIL can indicate termination of an **mm_Play( )** playback operation by **mm_Stop( )**. The event data provides result information about the reason for the completion or termination event.

Optional Notification Events

Format: **MMEV_xxxx** (where "xxxx" is the name of the enabled optional event)

These events are enabled by default and are enabled and disabled by the **mm_EnableEvents( )** and **mm_DisableEvents( )** functions. These events are generated upon encountering the condition specified by the event; for example, the MMEV_VIDEO_RECORD_STARTED optional **intermediate notification event** reports the the actual start of recording (detection of an I-frame or a time-out waiting for an I-frame). The MMEV_VIDEO_RECORD_STARTED_FAIL failure event that is associated with this operation is **reserved for future use** (there are no conditions that generate it).

Unsolicited Events

Format: **MMEV_xxxx** (where "xxxx" is the name of the unsolicited event)
These events are not requested by the application. They are triggered by, and provide information about, internal or external events. For example, MMEV_ERROR indicates an unexpected failure. The event data provides error result information.

Note: For more information on failure events, see Chapter 6, "Multimedia API Errors".

3.3 Dialogic® Multimedia API Event Types by Function Type

To identify the specific events that a particular function can generate, see the "Asynchronous Mode Events" section of that function under Chapter 2, "Function Information".

The following summarizes the type of events that different types of functions can generate:

- All asynchronous functions are subject to unsolicited events (e.g., MMEV_ERROR).
- All asynchronous functions can generate a successful termination/completion event or a failure termination event corresponding to the function operation (e.g., MMEV_PLAY and MMEV_PLAY_FAIL).

  Note: One exception is the **mm_Stop( )** function. Successful termination or completion is reported by MMEV_PLAY or MMEV_RECORD events, and failure is reported by MMEV_PLAY_FAIL or MMEV_RECORD_FAIL.
- All functions that initiate or terminate I/O, including **mm_Play( )**, **mm_Record( )**, **mm_Stop( )**, and **mm_Reset( )**, can generate a successful initiation event (which is an intermediate event) or a corresponding initiation failure termination event (e.g., MMEV_PLAY_ACK and MMEV_PLAY_ACK_FAIL).
- Only specific functions or operations generate optional notification events (e.g., MMEV_VIDEO_RECORD_STARTED applies only to **mm_Record( )** operation).

3.4 Dialogic® Multimedia API Events

The API can generate the following events (listed in alphabetical order):

MMEV_DISABLEEVENTS

Termination event reported upon successful completion of the function.

Event Data: MM_DISABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_DISABLEEVENTS_FAIL

Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_DISABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ENABLEEVENTS

Termination event reported upon successful completion of the function.

Event Data: MM_ENABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ENABLEEVENTS_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_ENABLE_EVENTS_RESULT structure (MM_RET_CODE)

MMEV_ERROR
Unsolicited failure event reported upon encountering an unexpected failure.

Event Data: MM_ERROR_RESULT structure

MMEV_GETPARM
Termination event reported upon successful completion of the function.

Event Data: MM_GET_PARM_RESULT structure

MMEV_GETPARM_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_GET_PARM_RESULT structure

MMEV_OPEN
Termination event reported upon successful completion of the function.

Event Data: MM_OPEN_RESULT structure (MM_RET_CODE)

MMEV_OPEN_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_OPEN_RESULT structure (MM_RET_CODE)

Note: The application program must call **mm_Close( )** to clean up after this failure.

MMEV_PLAY
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_PLAY_ACK
Initiation event reported upon successful start of the function.

Event Data: MM_PLAY_ACK structure (MM_RET_CODE)

MMEV_PLAY_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_PLAY_ACK structure (MM_RET_CODE)

MMEV_PLAY_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_PLAY_CMPLT structure (MM_PLAY_RECORD_CMPLT)

Note: See Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for related information.

MMEV_RECORD
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

MMEV_RECORD_ACK
Initiation event reported upon successful start of the function.

Event Data: MM_RECORD_ACK structure (MM_RET_CODE)

MMEV_RECORD_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_RECORD_ACK structure (MM_RET_CODE)

MMEV_RECORD_FAIL
Operation failure termination event indicating operation errors reported upon completion or termination of the operation. The operation errors that can cause this failure event are **intermediate (non-terminating) errors**.

Event Data: MM_RECORD_CMPLT structure (MM_PLAY_RECORD_CMPLT)

Note: See Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for related information.

MMEV_RESET
Termination event reported upon successful completion or successful termination of the operation.

Event Data: MM_RESET_RESULT structure (MM_RET_CODE)

MMEV_RESET_ACK
Initiation event reported upon successful start of the function.

Event Data: MM_RESET_ACK structure (MM_RET_CODE)

MMEV_RESET_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_RESET_ACK structure (MM_RET_CODE)

MMEV_RESET_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_RESET_RESULT structure (MM_RET_CODE)

MMEV_SETPARM
Termination event reported upon successful completion of the function.

Event Data: MM_SET_PARM_RESULT structure (MM_RET_CODE)

MMEV_SETPARM_FAIL
Operation failure termination event reported upon encountering an error during the operation.

Event Data: MM_SET_PARM_RESULT structure (MM_RET_CODE)

MMEV_STOP_ACK
Initiation event reported upon successful start of the function.

Event Data: MM_STOP_ACK structure

MMEV_STOP_ACK_FAIL
Initiation failure termination event reported upon encountering an error before the operation begins.

Event Data: MM_STOP_ACK structure

MMEV_VIDEO_RECORD_STARTED

Optional **intermediate (non-terminating) notification event** (enabled by default), indicating actual start of recording (complete video frame, or I-frame, received, or time-out waiting for an I-frame). This event can be disabled by **mm_DisableEvents( )** and enabled by **mm_EnableEvents( )**.

Event Data: MM_VIDEO_RECORD_STARTED structure

MMEV_VIDEO_RECORD_STARTED_FAIL (**reserved for future use**)

Optional **intermediate (non-terminating) failure notification event** that is **reserved for future use** (there are no conditions that generate it).

Data Structure Types 4

This chapter describes the *types* of data structures supported by the Dialogic® Multimedia API, including the basic categories, naming convention for specific types, their purpose, hierarchy or nesting levels, and association with specific functions.

- Overview of Dialogic® Multimedia API Data Structures. 53
- Data Structures for Function I/O 54
- Data Structures for Analyzing Event Information. 55
- Play/Record Data Structure Levels 57
- Other Data Structure Levels (_DETAILS) 57
- Data Structures By Function 58

Note: For an alphabetical list of Dialogic® Multimedia API data structures followed by a comprehensive reference organized alphabetically and providing detailed information on the structures, see Chapter 5, "Data Structure Reference".

4.1 Overview of Dialogic® Multimedia API Data Structures

The Dialogic Multimedia API data structures are defined in the *mmlib.h* header file. These data structures are used to control the operation of functions and to obtain information about events.

Multimedia API Data Structure Categories

There are two basic categories relating to how data structures are used in the Dialogic Multimedia API:

Data Structures for Function I/O

This category of data structure is used directly by the function as either **input** or **output**, depending upon the function. For example, the **pStop** parameter in the **mm_Stop( )** function points to an MM_STOP data structure that is used for input to the function.

A data structure may also be *nested* within the top level function I/O data structure. For example, the MM_STOP input data structure contains a details field that is a data structure of type MM_STOP_DETAILS, which is thus a 2nd level in the hierarchy. Additional *nesting levels* in data structure hierarchy are possible, as in the **mm_Play( )** and **mm_Record( )** functions, which reach up to 6 levels (see Section 4.4, "Play/Record Data Structure Levels", on page 57). See also Section 4.5, "Other Data Structure Levels (_DETAILS)", on page 57.

Data Structures for Analyzing Event Information

This category of data structure is used to analyze Multimedia API event data provided by the **mm_GetMetaEvent( )** function in the MM_METAEVENT data structure eventdatap field. For example, **mm_GetMetaEvent( )** function gets the event data for the MMEV_ENABLE_EVENTS or MMEV_ENABLE_EVENTS_FAIL function completion

events through an MM_ENABLE_EVENTS_RESULT (MM_RET_CODE) data structure, which provides result information on the function termination.

A data structure may also be *nested* within the top level structure of event data. For example, the MM_PLAY_RECORD_CMPLT structure for event data from a play or record termination event contains a details field that is a data structure of type MM_PLAY_RECORD_CMPLT_DETAILS, which is thus a second level in the hierarchy. Additional *nesting levels* in this type of information hierarchy are possible but are typically unnecessary.

4.2 Data Structures for Function I/O

The following information is a guide to understanding the naming convention for, and purpose of, data structures that are used for *function input or output*. This information is organized by the type of data structure, as identified in the name, usually through a suffix. In the following list, a name is given for a data structure type, where "xxxx" represents the name of the function, operation, or event which applies to the data structure.

MM_xxxx_**INFO** (function input/output information)

This type of data structure is used *primarily* for function **input**, but in some cases may be used for function output. For example, the MM_INFO data structure is used for **output** by the **mm_ErrorInfo( )** function. The following data structures are included in this type:

- MM_INFO – The **pInfo** parameter in the **mm_ErrorInfo( )** function points to an MM_INFO data structure that is used for **output** by the functions.
- MM_PLAY_INFO – Used as function input for **mm_Play( )**, it is of type MM_PLAY_RECORD_INFO.
- MM_PLAY_RECORD_INFO – This data structure is used for input by the **mm_Play( )** and **mm_Record( )** functions, and it contains data structures nested in a multi-level hierarchy within it. For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57. It is used as a typedef for the MM_RECORD_INFO and MM_RECORD_INFO structures.
- MM_RECORD_INFO – Used as function input for **mm_Record( )**, it is of type MM_PLAY_RECORD_INFO.

MM_**xxxx** (function input/output)

This type of data structure is used *primarily* for function **input**, but in some cases may be used for function output. For example, the MM_METAEVENT data structure is used for **output** by the **mm_GetMetaEvent( )** function. The following data structures are included in this type:

- MM_EVENTS – The **pEvents** parameter in the **mm_EnableEvents( )** function and the **mm_DisableEvents( )** function points to an MM_EVENTS data structure that is used for **input** by the functions.
- MM_METAEVENT – The **pMetaEvent** parameter in the **mm_GetMetaEvent( )** function points to an MM_METAEVENT data structure that is used for **output** by **mm_GetMetaEvent( )**.
- MM_GET_PARM – Used as function input for **mm_GetParm( )**.
- MM_SET_PARM – Used as function input for **mm_SetParm( )**.
- MM_STOP – Used as function input for **mm_Stop( )**. It contains a second-level MM_STOP_DETAILS structure that is **reserved for future use**.

4.3 Data Structures for Analyzing Event Information

Note: See Chapter 3, "Events" for additional information on events and event data.

The following information is a guide to understanding the naming convention for, and purpose of, data structures that are used for *analyzing event information*. This information is organized by the type of data structure, as identified in the name, usually through a suffix. In the following list, a name is given for a data structure type, where "xxxx" represents the name of the function, operation, or event which applies to the data structure.

MM_xxxx_**ACK** (function initiation event data)

This type of data structure is used for analyzing the results of a *function initiation event*. For example, the MM_PLAY_ACK data structure is used to analyze the result of an **mm_Play( )** function initiation as reported by the MMEV_PLAY_ACK or MMEV_PLAY_ACK_FAIL event. The following data structures are included in this type:

- MM_PLAY_ACK (MM_RET_CODE)*
- MM_RECORD_ACK (MM_RET_CODE)*
- MM_RESET_ACK (MM_RET_CODE)*
- MM_STOP_ACK
- MM_STOP_ACK_DETAILS

Note: The MM_STOP_ACK_DETAILS data structure is a second-level event data structure under the MM_STOP_ACK structure.

*These structures are of type MM_RET_CODE.

MM_xxxx_**CMPLT** (media function termination event data)

This type of data structure is used for analyzing the results of a *media function termination or completion event*. For example, the MM_PLAY_CMPLT data structure is used to analyze the result of an **mm_Play( )** function as reported by the MMEV_PLAY or MMEV_PLAY_FAIL event. The following data structures are included in this type:

- MM_PLAY_CMPLT (MM_PLAY_RECORD_CMPLT)*
- MM_PLAY_RECORD_CMPLT
- MM_RECORD_CMPLT (MM_PLAY_RECORD_CMPLT)*
- MM_PLAY_RECORD_CMPLT_DETAILS

Note: The MM_PLAY_RECORD_CMPLT_DETAILS data structure is a second-level event data structure under the MM_PLAY_RECORD_CMPLT structure.

*These structures are of type MM_PLAY_RECORD_CMPLT.

MM_xxxx_**DETAILS** (additional event data details)

This type of data structure is used for analyzing *additional details of event information*. For example, the MM_PLAY_RECORD_CMPLT_DETAILS data structure is used to analyze the result of an **mm_Play( )** or **mm_Record( )** function as reported by the MMEV_PLAY, MMEV_PLAY_FAIL, MMEV_RECORD, or MMEV_RECORD_FAIL event. The following data structures are included in this type (these data structures encompass a sub-type because they provide details for a higher-level structure; see also Section 4.5, "Other Data Structure Levels (_DETAILS)", on page 57):

- MM_STOP_ACK_DETAILS

Note: The MM_STOP_ACK_DETAILS data structure is a second-level event data structure under the MM_STOP_ACK structure.

- MM_PLAY_RECORD_CMPLT_DETAILS

Note: The MM_PLAY_RECORD_CMPLT_DETAILS data structure is a second-level event data structure under the MM_PLAY_RECORD_CMPLT structure.

MM_xxxx_**RESULT** (results of a termination or completion event)

This type of data structure is used for analyzing the *results of a termination or completion event*, primarily non-media function operation events. For example, the MM_ENABLE_EVENTS_RESULT data structure is used to analyze the result of an **mm_EnableEvents( )** function as reported by the MMEV_ENABLEEVENTS or MMEV_ENABLEEVENTS_FAIL event. The following data structures are included in this type:

- MM_DISABLE_EVENTS_RESULT (MM_RET_CODE)*
- MM_ENABLE_EVENTS_RESULT (MM_RET_CODE)*
- MM_ERROR_RESULT

Note: The MM_ERROR_RESULT data structure is used to analyze the causes of an MMEV_ERROR unsolicited failure event.

- MM_GET_PARM_RESULT
- MM_OPEN_RESULT (MM_RET_CODE)*
- MM_RESET_RESULT (MM_RET_CODE)*
- MM_SET_PARM_RESULT (MM_RET_CODE)*

*These structures are of type MM_RET_CODE

MM_**RET_CODE** (error information event data structure)

The MM_RET_CODE event information data structure describes error return code information related to an event. (See Section 6.3, "Dialogic® Multimedia API Event Information Error Return Codes", on page 96 for a list of the error return codes and related information.) This structure is used as a typedef for the following event information data structures.

ACK event information structures:

1. MM_PLAY_ACK
2. MM_RECORD_ACK
3. MM_RESET_ACK

RESULT event information structures:

4. MM_DISABLE_EVENTS_RESULT
5. MM_ENABLE_EVENTS_RESULT
6. MM_OPEN_RESULT
7. MM_RESET_RESULT
8. MM_SET_PARM_RESULT

Note: Error codes are also returned in the MM_STOP_ACK_DETAILS unRetCode field, which is used to provide details on the MM_STOP_ACK event data associated with the MMEV_STOP_ACK and MMEV_STOP_ACK_FAIL events.

4.4 Play/Record Data Structure Levels

The following hierarchy (starting at the function parameter level) illustrates the *nesting levels* for the function input data structures used by the **mm_Play( )** and **mm_Record( )** functions. The hierarchy is shown here by indenting the nested data structures and indicating the nesting level with a level number.

```
mm_Play( )( nDeviceHandle, pPlayInfo, pRuntimeControl, pUserInfo )
mm_Record( )( nDeviceHandle, pRecordInfo, pRuntimeControl, pUserInfo )

The pPlayInfo and pRecordInfo parameters point to the first level data structure.

1. MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO)*

     2. MM_PLAY_RECORD_LIST list

          3. MM_MEDIA_ITEM_LIST list

               4. MM_MEDIA_ITEM item

                    5. MM_MEDIA_VIDEO video

                         6. MM_VIDEO_CODEC codec

                    5. MM_MEDIA_AUDIO audio

                         6. MM_AUDIO_CODEC codec

                    5. MM_MEDIA_TERM term  [reserved for future use]
```

The MM_PLAY_INFO structure pointed to by **pPlayInfo, and the MM_RECORD_INFO structure pointed to by **pRecordInfo**, are both of typedef MM_PLAY_RECORD_INFO.*

4.5 Other Data Structure Levels (_DETAILS)

The following data structures also contain a hierarchy, or *nesting level*s, but they are limited to a more simple, ***two-level hierarchy***:

- The MM_PLAY_RECORD_CMPLT_DETAILS data structure is a second-level event information data structure under the MM_PLAY_RECORD_CMPLT (MM_PLAY_CMPLT, MM_RECORD_CMPLT) structure. These event information data structures are used for analyzing the results of a *media function termination or completion event* (MMEV_PLAY, MMEV_PLAY_FAIL, MMEV_RECORD, and MMEV_RECORD_FAIL). Both MM_PLAY_CMPLT and MM_RECORD_CMPLT are of typedef MM_PLAY_RECORD_CMPLT.
- The MM_STOP_ACK_DETAILS data structure is a second-level event information data structure under the MM_STOP_ACK structure. These event information data structures are used for analyzing the results of the MMEV_STOP_ACK or MMEV_STOP_ACK_FAIL function initiation events produced by the **mm_Stop( )** function.
- The MM_STOP_DETAILS data structure (**reserved for future use**) is a second-level function input data structure under the MM_STOP structure. These data structures provide input for the **mm_Stop( )** function.

4.6 Data Structures By Function

The following list identifies the data structures associated with specific Dialogic Multimedia API functions, either as input or through event information. Some of the data structures are associated with more than one function.

Note: For an alphabetical list of Dialogic Multimedia API data structures followed by a comprehensive reference organized alphabetically and providing detailed information on the structures, see Chapter 5, "Data Structure Reference".

The parenthetical codes in the following list represent the type of structure: **E** for *event information*, **FI** for *function input*, **FO** for f*unction output*, and **#** (a number) for nesting level when the structure occurs in a hierarchy. For example, FI2 indicates a second-level function input structure.

The parenthetical codes in the following list represent the type of structure:

- **FI** for *function input*,
- **FO** for f*unction output*,
- **E** for *event information*,
- **L#** for *nesting level number* (when the structure occurs in a hierarchy)
  For example, FI L2 indicates a function input second-level structure.

All Asynchronous Functions

- MM_ERROR_RESULT (E)

mm_DisableEvents( )

- MM_EVENTS (FI)
- MM_DISABLE_EVENTS_RESULT (MM_RET_CODE) (EI)

mm_EnableEvents( )

- MM_EVENTS (FI)
- MM_ENABLE_EVENTS_RESULT (MM_RET_CODE) (EI)

mm_ErrorInfo( )

- MM_INFO (FO)

mm_GetMetaEvent( )

- MM_METAEVENT (FO)

mm_GetParm( )

- MM_GET_PARM (FI)
- MM_GET_PARM_RESULT (E)

mm_Open( )

- MM_OPEN_RESULT (MM_RET_CODE) (E)

mm_Play( )

- MM_PLAY_INFO (MM_PLAY_RECORD_INFO) (FI L1)
- MM_PLAY_RECORD_LIST (FI L2)
- MM_MEDIA_ITEM_LIST (FI L3)
- MM_MEDIA_ITEM (FI L4)
- MM_MEDIA_VIDEO (FI L5)
- MM_VIDEO_CODEC (FI L6)
- MM_MEDIA_AUDIO (FI L5)
- MM_AUDIO_CODEC (FI L6)
- MM_MEDIA_TERM (FI L5) (**reserved for future use**)
- MM_PLAY_ACK (MM_RET_CODE) (E)
- MM_PLAY_CMPLT (MM_PLAY_RECORD_CMPLT) (E L1)
- MM_PLAY_RECORD_CMPLT_DETAILS (E L2)

mm_Record( )

- MM_RECORD_INFO (MM_PLAY_RECORD_INFO) (FI L1)
- MM_PLAY_RECORD_LIST (FI L2)
- MM_MEDIA_ITEM_LIST (FI L3)
- MM_MEDIA_ITEM (FI L4)
- MM_MEDIA_VIDEO (FI L5)
- MM_VIDEO_CODEC (FI L6)
- MM_MEDIA_AUDIO (FI L5)
- MM_AUDIO_CODEC (FI L6)
- MM_MEDIA_TERM (FI L5) (**reserved for future use**)
- MM_RECORD_ACK (MM_RET_CODE) (E)
- MM_RECORD_CMPLT (MM_PLAY_RECORD_CMPLT) (E L1)
- MM_PLAY_RECORD_CMPLT_DETAILS (E L2)
- MM_VIDEO_RECORD_STARTED (E)

mm_Reset( )

- MM_RESET_ACK (MM_RET_CODE) (E)
- MM_RESET_RESULT (MM_RET_CODE) (E)
- MM_PLAY_RECORD_CMPLT) (E L1)
- MM_PLAY_RECORD_CMPLT_DETAILS (E L2)

mm_SetParm( )

- MM_SET_PARM (FI)
- MM_SET_PARM_RESULT (MM_RET_CODE) (E)

mm_Stop( )

- MM_STOP (FI)
- MM_STOP_DETAILS (FI L2) (**reserved for future use**)
- MM_STOP_ACK (E L1)
- MM_STOP_ACK_DETAILS (E L2)
- MM_PLAY_RECORD_CMPLT) (E L1)
- MM_PLAY_RECORD_CMPLT_DETAILS (E L2)

Data Structure Reference 5

This chapter provides a list of data structures followed by a comprehensive data structure reference organized alphabetically and providing detailed information on structures.

Note: See Chapter 4, "Data Structure Types" for a description of the *types* of data structures supported by the Dialogic® Multimedia API library, including the basic categories, naming convention for specific types, their purpose, hierarchy or nesting levels, and association with specific functions.

- MM_AUDIO_CODEC 63
- MM_ERROR_RESULT 64
- MM_EVENTS 65
- MM_GET_PARM 66
- MM_GET_PARM_RESULT 67
- MM_INFO 68
- MM_MEDIA_AUDIO 69
- MM_MEDIA_ITEM 70
- MM_MEDIA_ITEM_LIST 71
- MM_MEDIA_TERM 72
- MM_MEDIA_VIDEO 73
- MM_METAEVENT 74
- MM_PLAY_RECORD_CMPLT 76
- MM_PLAY_RECORD_CMPLT_DETAILS 77
- MM_PLAY_RECORD_INFO 79
- MM_PLAY_RECORD_LIST 80
- MM_RET_CODE 82
- MM_SET_PARM 84
- MM_STOP 85
- MM_STOP_ACK 87
- MM_STOP_ACK_DETAILS 88
- MM_STOP_DETAILS 89
- MM_VIDEO_CODEC 90
- MM_VIDEO_RECORD_STARTED 92

Note: The following data structures are not listed under their own name but under the name of their typedef structure. The following list shows the type followed by a list of structure names that are of that type:

MM_PLAY_RECORD_INFO
- MM_PLAY_INFO
- MM_RECORD_INFO

MM_PLAY_RECORD_CMPLT
- MM_PLAY_CMPLT
- MM_RECORD_CMPLT

MM_RET_CODE
- MM_PLAY_ACK
- MM_RECORD_ACK
- MM_RESET_ACK

MM_RET_CODE
- MM_DISABLE_EVENTS_RESULT
- MM_ENABLE_EVENTS_RESULT
- MM_OPEN_RESULT
- MM_RESET_RESULT
- MM_SET_PARM_RESULT

MM_AUDIO_CODEC

```
typedef struct tagMM_AUDIO_CODEC
{
    unsigned int     unVersion;
    unsigned int     unCoding;
    unsigned int     unSampleRate;
    unsigned int     unBitsPerSample;
} MM_AUDIO_CODEC, *PMM_AUDIO_CODEC;
typedef const MM_AUDIO_CODEC* CPMM_AUDIO_CODEC
```

Description

The MM_AUDIO_CODEC structure specifies the characteristics of the audio coder. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_AUDIO structure, and it is a sixth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Sets the version of the structure (where "v" is the version number).

  Use MM_AUDIO_CODEC_VER(v) macro to set the version of the structure.

  Use MM_AUDIO_CODEC_VERSION_v macro to set to version v.

unCoding
: Specifies the type of audio coding. Defined values include:
  - MM_DATA_FORMAT_PCM (or Voice API: DATA_FORMAT_PCM) – Linear PCM, mono, LSB-MSB ("little endian")

unSampleRate
: Specifies the audio sampling rate in samples per second. Defined values include:
  - MM_DRT_8KHZ (or Voice API: DRT_8KHZ) – 8000

unBitsPerSample
: Specifies the number of bits per audio sample. Defined values include:
  - 16

Note: The Dialogic® Multimedia API library supports the Linear PCM (128 kbps) audio file format. This audio file format can also be used in the Dialogic® Voice API library by setting the DX_XPB structure fields to the following values:

- wFileFormat = FILE_FORMAT_VOX
- wDataFormat = DATA_FORMAT_PCM
- nSamplesPerSecond = DRT_8KHZ
- wBitsPerSample = 16

MM_ERROR_RESULT

```
typedef struct tagMM_ERROR_RESULT
{
    unsigned int    unVersion;
    unsigned int    unErrorCode;
    unsigned int    unErrorMsg;
    unsigned int    unData[4];
} MM_ERROR_RESULT, *PMM_ERROR_RESULT;
typedef const MM_ERROR_RESULT* CPMM_ERROR_RESULT;
```

Description

The MM_ERROR_RESULT *event information* data structure is used for analyzing the MMEV_ERROR unsolicited failure event.

If the MMEV_ERROR unsolicited failure event occurs, use the **mm_GetMetaEvent( )** function to retrieve the reason for the error. The **mm_GetMetaEvent( )** function outputs the MM_ERROR_RESULT event data associated with the metaevent in the MM_METAEVENT data structure.

For a list of the error codes and messages returned in the MM_INFO structure, see Section 6.2, “Dialogic® Multimedia API Function Error Codes”, on page 95.

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Provides the version of the structure (where “v” is the version number).

  Use MM_ERROR_RESULT_VER(v) macro to check the version of the structure.

  Use MM_ERROR_RESULT_VERSION_v macro to check version v.

unErrorCode
: Specifies an error code. See Section 6.2, “Dialogic® Multimedia API Function Error Codes”, on page 95 for the possible values.

unErrorMsg
: Specifies an error message associated with the code. See Section 6.2, “Dialogic® Multimedia API Function Error Codes”, on page 95 for the values.

unData
: Specifies an internal block of firmware data for debugging information related to the error; application program can record it in an error log along with the other fields.

MM_EVENTS

```
typedef struct tagMM_EVENTS
{
    unsigned int    unVersion;
    unsigned int    unMask;
} MM_EVENTS, *PMM_EVENTS;
typedef const MM_EVENTS* CPMM_EVENTS;
```

Description

The MM_EVENTS structure specifies the details of an enable or disable events request. This structure is used as *function input* for the **mm_EnableEvents( )** and **mm_DisableEvents( )** functions.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_EVENTS_VER(v) macro to set the version of the structure.

Use MM_EVENTS_VERSION_v macro to set to version v.

unMask

Specifies event mask to enable or disable. The mask is formed from one or more of the following:

- MMR_EVENT_VIDEO_RECORD_STARTED – Bitmask to control MMEV_VIDEO_RECORD_STARTED event.

Note: See **mm_SetParm( )** and the MM_SET_PARM structure for information on the related EMM_REC_IFRAME_TIMEOUT parameter. See also the **mm_Record( )** function.

MM_GET_PARM

```
typedef struct tagMM_GET_PARM
{
    unsigned int    unVersion;
    eMM_PARM        eParm;
} MM_GET_PARM, *PMM_GET_PARM;
typedef const MM_GET_PARM* CPMM_GET_PARM;
```

Description

The MM_GET_PARM structure specifies the details of a get parameter request. This structure is used as *function input* for the **mm_GetParm( )** function.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_GET_PARM_VER(v) macro to set the version of the structure.

Use MM_GET_PARM_VERSION_v macro to set to version v.

eParm

Specifies the parameter to get. Defined values include the following channel-level parameter:

- EMM_REC_IFRAME_TIMEOUT – Specifies the time to wait for an I-frame. Video recording, or multimedia (audio and video) recording, starts when an I-frame is detected or when the time-out is reached.

MM_GET_PARM_RESULT

```
typedef struct tagMM_GET_PARM_RESULT
{
    unsigned int    unVersion;
    eMM_PARM        eParm;
    unsigned int    unParmValue;
} MM_GET_PARM_RESULT, *PMM_GET_PARM_RESULT;
typedef const MM_GET_PARM_RESULT* CPMM_GET_PARM_RESULT;
```

Description

The MM_GET_PARM_RESULT *event information* data structure is used for analyzing the results of the **mm_GetParm( )** function **termination or completion**, as reported by the MMEV_GET_PARM or MMEV_GET_PARM_FAIL event.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Provides the version of the structure (where "v" is the version number).

Use MM_GET_PARM_RESULT_VER(v) macro to check the version of the structure.

Use MM_GET_PARM_RESULT_VERSION_v macro to check version v.

eParm

Specifies parameter requested. Defined values include the following channel-level parameter:

- EMM_REC_IFRAME_TIMEOUT – Specifies the time to wait for an I-frame. Video recording, or multimedia (audio and video) recording, starts when an I-frame is detected or when the time-out is reached.

unParmValue

Parameter value. See MM_SET_PARM, on page 84 for the values.

MM_INFO

```
typedef struct tagMM_INFO
{
    unsigned int  unVersion;
    int           mmValue;
    const char*   mmMsg;
    const char*   additionalInfo;
} MM_INFO, *PMM_INFO;
typedef const MM_INFO* CPMM_INFO;
```

Description

The MM_INFO structure is used as *function output* for the **mm_ErrorInfo( )** function and contains error or result information.

Field Descriptions

The fields of MM_INFO are described as follows:

unVersion
: Sets the version of the structure (where "v" is the version number).

  Use MM_INFO_VER(v) macro to set the version of the structure.

  Use MM_INFO_VERSION_v macro to set to version v.

mmValue
: Specifies a Dialogic® Multimedia API error or result value. See Section 6.2, "Dialogic® Multimedia API Function Error Codes", on page 95.

mmMsg
: Specifies a pointer to a null-terminated string containing a message associated with the Multimedia API error or result value. See Section 6.2, "Dialogic® Multimedia API Function Error Codes", on page 95.

additionalInfo
: Specifies a pointer to a null-terminated string containing additional information associated with this error or result value. This additional information is optional and may be used as a diagnostic aid. .

MM_MEDIA_AUDIO

```
typedef struct tagMM_MEDIA_AUDIO
{
    unsigned int     unVersion;
    MM_AUDIO_CODEC   codec;
    unsigned int     unMode;
    const char*      szFileName;
    unsigned int     unOffset;
} MM_MEDIA_AUDIO, *PMM_MEDIA_AUDIO;
typedef const CPMM_MEDIA_AUDIO* CPMM_MEDIA_AUDIO;
```

Description

The MM_MEDIA_AUDIO structure specifies the audio media item. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_ITEM structure, and it is a fifth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Sets the version of the structure (where "v" is the version number).

  Use MM_MEDIA_AUDIO_VER(v) macro to set the version of the structure.

  Use MM_MEDIA_AUDIO_VERSION_v macro to set to version v.

codec
: Specifies the audio codec. See definition of MM_AUDIO_CODEC structure.

unMode
: Specifies the mask for the mode of operation. The mask is formed from one or more of the following:

  - MM_MODE_AUD_BEEPINITIATED – Bitmask to **enable** notification tone (or "beep") at start of **audio play or record**.

    Note: If doing a multimedia playback or record with synchronized audio and video, only a single beep is generated even if the corresponding MM_MEDIA_VIDEO beep is enabled.

  - MM_MODE_AUD_FILE_TYPE_VOX – Bitmask to specify VOX audio file.

szFileName
: Specifies the file name of the audio media item.

unOffset
: This field is reserved for future use.

MM_MEDIA_ITEM

```
typedef union tagMM_MEDIA_ITEM
{
    unsigned int      unVersion;
    MM_MEDIA_VIDEO    video;
    MM_MEDIA_AUDIO    audio;
    MM_MEDIA_TERM     term;
} MM_MEDIA_ITEM, *PMM_MEDIA_ITEM;
typedef const MM_MEDIA_ITEM* CPMM_MEDIA_ITEM;
```

Description

The MM_MEDIA_ITEM union specifies the media item. This union is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_ITEM_LIST structure, and it is a fourth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The elements of the union are described as follows:

unVersion
: Sets the version of the structure (where "v" is the version number).

  Use MM_MEDIA_ITEM_VER(v) macro to set the version of the structure.

  Use MM_MEDIA_ITEM_VERSION_v macro to set to version v.

video
: Specifies the video item. See definition of MM_MEDIA_VIDEO structure.

audio
: Specifies the audio item. See definition of MM_MEDIA_AUDIO structure.

term
: Specifies the termination condition item. See definition of MM_MEDIA_TERM structure.

  Note: Although this structure is **reserved for future use**, you must pass the structure and set the unRFU field to 0.

MM_MEDIA_ITEM_LIST

```
typedef struct tagMM_MEDIA_ITEM_LIST
{
    unsigned int                    unVersion;
    eMM_ITEM                        ItemChain;
    MM_MEDIA_ITEM                   item;
    struct tagMM_MEDIA_ITEM_LIST*   next;
    struct tagMM_MEDIA_ITEM_LIST*   prev;   /* optional */
} MM_MEDIA_ITEM_LIST, *PMM_MEDIA_ITEM_LIST;
typedef const MM_MEDIA_ITEM_LIST* CPMM_MEDIA_ITEM_LIST;
```

Description

The MM_MEDIA_ITEM_LIST structure specifies the media item list. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_PLAY_RECORD_LIST structure, and it is a third-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_MEDIA_ITEM_LIST_VER(v) macro to set the version of the structure.

Use MM_MEDIA_ITEM_LIST_VERSION_v macro to set to version v.

ItemChain

Specifies the next list element for iteration. Defined values include:

- EMM_ITEM_CONT – Indicates that the next list element immediately follows the current.
- EMM_ITEM_LINK – Indicates that the next and previous list elements linked to the current via pointers.
- EMM_ITEM_EOT – Indicates that the current list element is the last in the list.

item

Specifies the media item. See definition of MM_MEDIA_ITEM structure.

next

Points to the next MM_MEDIA_ITEM_LIST item. Needed only when ItemChain specifies a linked list element. NULL value indicates last item.

prev

Points to the previous MM_MEDIA_ITEM_LIST item (optional). Needed only when ItemChain specifies a linked list element. NULL value indicates first item.

MM_MEDIA_TERM

```
typedef struct tagMM_MEDIA_TERM
{
    unsigned int   unVersion;
    unsigned int   unRfu;
} MM_MEDIA_TERM, *PMM_MEDIA_TERM;
typedef const MM_MEDIA_TERM* CPMM_MEDIA_TERM;
```

Description

Note: Although this structure is **reserved for future use**, you must pass the structure and set the unRFU field to 0.

The MM_MEDIA_TERM structure specifies termination details. It is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_ITEM union, and it is a fifth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_MEDIA_TERM_VER(v) macro to set the version of the structure.

Use MM_MEDIA_TERM_VERSION_v macro to set to version v.

unRFU

Reserved for future use. Must be set to 0.

MM_MEDIA_VIDEO

```
typedef struct tagMM_MEDIA_VIDEO
{
    unsigned int      unVersion;
    MM_VIDEO_CODEC    codec;
    unsigned int      unMode;
    const char*       szFileName;
} MM_MEDIA_VIDEO, *PMM_MEDIA_VIDEO;
typedef const MM_MEDIA_VIDEO* CPMM_MEDIA_VIDEO;
```

Description

The MM_MEDIA_VIDEO structure specifies the video media item. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_ITEM structure, and it is a fifth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Sets the version of the structure (where "v" is the version number).

  Use MM_MEDIA_VIDEO_VER(v) macro to set the version of the structure.

  Use MM_MEDIA_VIDEO_VERSION_v macro to set to version v.

codec
: Specifies the video codec. See definition of MM_VIDEO_CODEC structure.

unMode
: Specifies the mask for the mode of operation. The mask is formed from one or more of the following:

  - MM_MODE_VID_BEEPINITIATED – Bitmask to **enable** notification tone (or "beep") at start of **video play**.
  - MM_MODE_VID_NOIFRMBEEPINITIATED – Bitmask to **disable** notification tone at start of **video record**. (The record beep is enabled by default and is transmitted upon detection of an I-frame or upon time-out waiting for an I-frame.)

  Note: If doing a multimedia playback or record with synchronized audio and video, only a single beep is generated even if the corresponding MM_MEDIA_AUDIO beep is enabled.

szFileName
: Specifies the file name of the video media item.

MM_METAEVENT

```
typedef struct tagMM_METAEVENT
{
    unsigned int     unVersion;
    unsigned long    flags;
    void*            evtdatap;
    long             evtlen;
    long             evtdev;
    long             evttype;
    void*            evtUserInfo;
    int              rful;
} MM_METAEVENT, *PMM_METAEVENT;
typedef const MM_METAEVENT* CPMM_METAEVENT;
```

Description

The MM_METAEVENT structure is used as *function output* for the **mm_GetMetaEvent( )** function.

The MM_METAEVENT structure contains the event descriptor for a metaevent. The Field Descriptions section below describes each element used in the metaevent data structure, and where applicable, indicates the Dialogic® Standard Runtime Library (SRL) API function that is used to retrieve the information stored in the associated field. This data structure eliminates the need for the application to issue the equivalent SRL functions listed below. For information on the Dialogic SRL API, see the *Dialogic® Standard Runtime Library API Library Reference*.

Field Descriptions

The fields of MM_METAEVENT are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_METAEVENT_VER(v) macro to set the version of the structure.

Use MM_METAEVENT_VERSION_v macro to set to version v.

flags

Specifies any flags for the information in the structure. If the metaevent is a Dialogic® Multimedia API event, the MMME_MM_EVENT bit in the flags field will be set. The MM_METAEVENT fields contain valid Dialogic Multimedia API-related data only when the MMME_MM_EVENT bit is set. Do **not** use these fields for obtaining multimedia information if the bit is not set.

evtdatap

Specifies a pointer to the data associated with the event

Function equivalent: **sr_getevtdatap( )**

evtlen

Specifies the length of the data associated with the event

Function equivalent: **sr_getevtlen( )**

evtdev
: Specifies the event device

  Function equivalent: **sr_getevtdev( )**

evttype
: Event type

  Function equivalent: **sr_getevttype( )**

evtUserInfo
: Points to the user information associated with the event. This is the data passed by a **pUserInfo** parameter in an asynchronous Multimedia API function call. If no data was passed, it points to NULL.

  Function equivalent: **sr_getUserContext( )**

rfu1
: **Reserved for future use.** Must be set to 0.

MM_PLAY_RECORD_CMPLT

```
typedef struct tagMM_PLAY_RECORD_CMPLT
{
    unsigned int                    unVersion;
    unsigned int                    unCount;
    MM_PLAY_RECORD_CMPLT_DETAILS    details[MAX_PLAY_RECORD_CMPLT];
} MM_PLAY_RECORD_CMPLT, *PMM_PLAY_RECORD_CMPLT;
typedef const MM_PLAY_RECORD_CMPLT* CPMM_PLAY_RECORD_CMPLT;

typedef MM_PLAY_RECORD_CMPLT MM_PLAY_CMPLT, *PMM_PLAY_CMPLT;
typedef CPMM_PLAY_RECORD_CMPLT CPMM_PLAY_CMPLT;

typedef MM_PLAY_RECORD_CMPLT MM_RECORD_CMPLT, *PMM_RECORD_CMPLT;
typedef CPMM_PLAY_RECORD_CMPLT CPMM_RECORD_CMPLT;
```

Description

The MM_PLAY_RECORD_CMPLT *event information* data structure is used for analyzing the results of the **mm_Play( )** or **mm_Record( )** function **termination or completion** as reported by an MMEV_PLAY, MMEV_PLAY_FAIL, MMEV_RECORD, or MMEV_RECORD_FAIL event.

Note: This structure is used as a typedef for the MM_PLAY_CMPLT and MM_RECORD_CMPLT *event information* structures, which are defined as a convenience.

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Provides the version of the structure (where "v" is the version number).

  Use MM_PLAY_RECORD_CMPLT_VER(v) macro to check the version of the structure.

  Use MM_PLAY_RECORD_CMPLT_VERSION_v macro to check version v.

unCount
: Specifies the number of MM_PLAY_RECORD_CMPLT_DETAIL elements that follow.

details
: Specifies the details of completion. See MM_PLAY_RECORD_CMPLT_DETAILS structure definition for details.

  Note: See Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for related information.

MM_PLAY_RECORD_CMPLT_DETAILS

```
typedef struct tagMM_PLAY_RECORD_CMPLT_DETAILS
{
    unsigned int                    unVersion;
    eMM_CMPLT_PLAY_RECORD           Complete;
    eMM_CMPLT_PLAY_RECORD_REASON    Reason;
    unsigned int                    unDuration;
    unsigned int                    unNumberOfBytes;
    eMM_CMPLT_PLAY_RECORD_STATUS    Status;
} MM_PLAY_RECORD_CMPLT_DETAILS, *PMM_PLAY_RECORD_CMPLT_DETAILS;
typedef const MM_PLAY_RECORD_CMPLT_DETAILS* CPMM_PLAY_RECORD_CMPLT_DETAILS;
```

Description

The MM_PLAY_RECORD_CMPLT_DETAILS structure specifies the elements of details of MMEV_PLAY, MMEV_PLAY_FAIL, MMEV_RECORD, or MMEV_RECORD_FAIL events. This structure is a second-level *event information* data structure under the MM_PLAY_RECORD_CMPLT structure. (See Section 4.5, "Other Data Structure Levels (_DETAILS)", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Provides the version of the structure (where "v" is the version number).

  Use MM_PLAY_RECORD_CMPLT_DETAILS_VER(v) macro to check the version of the structure.

  Use MM_PLAY_RECORD_CMPLT_DETAILS_VERSION_v macro to check version v.

Complete
: Specifies the type of terminated or completed operation. Defined values include:
  - EMM_CMPLT_AUDIO_PLAY – Audio playback operation.
  - EMM_CMPLT_AUDIO_RECORD – Audio recording operation.
  - EMM_CMPLT_VIDEO_PLAY – Video playback operation.
  - EMM_CMPLT_VIDEO_RECORD – Video recording operation.

Reason
: Specifies the termination or completion reason. Defined values include:
  - EMM_TR_EOF – End of file on playback.
  - EMM_TR_ERROR – Terminating or non-terminating error. See Status field for details. See also Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97.
  - EMM_TR_USERSTOP – Stopped with the **mm_Stop( )** function.

unDuration
: Specifies the duration of play or record in milliseconds.

unNumberOfBytes
: Specifies the number of bytes played or recorded.

Status

Specifies any error during play or record. (See also Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97.) Defined values include:

- EMM_STATUS_PLAY_A_FILEREAD_ERROR – Audio file read error.
- EMM_STATUS_PLAY_V_ERROR_FS_GT_MFS – Video frame size greater than maximum frame size of internal buffer.
- EMM_STATUS_PLAY_V_FILEREAD_ERROR – Video file read error.
- EMM_STATUS_RCRD_A_DRPD_FRAME_FULL_ERROR – Full audio frame dropped error.
- EMM_STATUS_RCRD_V_DRPD_FRAME_FULL_ERROR – Full video frame dropped error.
- EMM_STATUS_RCRD_V_PKTS_DROPD_FS_GT_MFS – Video packets dropped frame size greater than maximum frame size of internal buffer.
- EMM_STATUS_SUCCESS – Successful completion or successful termination (no error).

MM_PLAY_RECORD_INFO

```
typedef struct tagMM_PLAY_RECORD_INFO
{
    unsigned int            unVersion;
    eMM_FILE_FORMAT         eFileFormat;
    CPMM_PLAY_RECORD_LIST   list;
} MM_PLAY_RECORD_INFO, *PMM_PLAY_RECORD_INFO;
typedef const MM_PLAY_RECORD_INFO* CPMM_PLAY_RECORD_INFO;

typedef MM_PLAY_RECORD_INFO MM_PLAY_INFO, *PMM_PLAY_INFO;
typedef CPMM_PLAY_RECORD_INFO CPMM_PLAY_INFO;

typedef MM_PLAY_RECORD_INFO MM_RECORD_INFO, *PMM_RECORD_INFO;
typedef CPMM_PLAY_RECORD_INFO CPMM_RECORD_INFO;
```

Description

This structure is a *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is a first-level structure that contains a nested hierarchy of structures under it. For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Note: This structure is used as a typedef for the MM_PLAY_INFO and MM_RECORD_INFO *function input* structures, which are defined as a convenience for use by the **mm_Play( )** and **mm_Record( )** functions.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_PLAY_RECORD_INFO_VER(v) macro to set the version of the structure.

Use MM_PLAY_RECORD_INFO_VERSION_v macro to set to version v.

eFileFormat

Specifies format of the files. Defined values include:

- EMM_FILE_FORMAT_PROPRIETARY

list

Points to the list of items to process. See definition of MM_PLAY_RECORD_LIST structure.

MM_PLAY_RECORD_LIST

```
typedef struct tagMM_PLAY_RECORD_LIST
{
    unsigned int                    unVersion;
    eMM_ITEM                        ItemChain;
    eMM_MEDIA_TYPE                  ItemType;
    CPMM_MEDIA_ITEM_LIST            list;
    unsigned int                    unRFU;
    struct tagMM_PLAY_RECORD_LIST*  next;
    struct tagMM_PLAY_RECORD_LIST*  prev;   /* optional */
} MM_PLAY_RECORD_LIST, *PMM_PLAY_RECORD_LIST;
typedef const MM_PLAY_RECORD_LIST* CPMM_PLAY_RECORD_LIST;
```

Description

The MM_PLAY_RECORD_LIST structure specifies the list of items to play or record. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is a second-level structure nested directly under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_PLAY_RECORD_LIST_VER(v) macro to set the version of the structure.

Use MM_PLAY_RECORD_LIST_VERSION_v macro to set to version v.

ItemChain

Specifies the next list element for iteration. Defined values include:

- EMM_ITEM_CONT – Indicates that the next list element immediately follows the current.
- EMM_ITEM_LINK – Indicates that the next and previous list elements linked to the current via pointers.
- EMM_ITEM_EOT – Indicates that the current list element is the last in the list.

ItemType

Specifies the type of the media item list. Defined values include:

- EMM_MEDIA_TYPE_VIDEO – Indicates that the current item specifies video element.
- EMM_MEDIA_TYPE_AUDIO – Indicates that the current item specifies audio element.

list

Points to the list of media items. See definition of MM_MEDIA_ITEM_LIST structure.

unRFU

Reserved for future use. Must be set to 0.

next

Points to the next MM_PLAY_RECORD_LIST item. Needed only when ItemChain specifies a linked list element. NULL value indicates last item.

prev

Points to the previous MM_PLAY_RECORD_LIST item (optional). Needed only when ItemChain specifies a linked list element. NULL value indicates first item.

MM_RET_CODE

```
typedef struct tagMM_RET_CODE
{
    unsigned int    unVersion;
    unsigned int    unRetCode;
} MM_RET_CODE, *PMM_RET_CODE;
typedef const MM_RET_CODE* CPMM_RET_CODE;
```

Description

The MM_RET_CODE *event information* data structure is used for analyzing the results of certain functions' **initiation** failure events (reported by their MMEV_xxxx_ACK_FAIL event) and certain functions' **termination or completion** failure events. Collectively, they are *non-media I/O operation failure termination events*. The event data for these metaevents contains an MM_RET_CODE structure with an error return code in the unRetCode field. These *event information error return codes* apply to any metaevent (provided by **mm_GetMetaEvent( )**) for which the event data is of type MM_RET_CODE.

Note: The MM_RET_CODE structure is used as a typedef for the following event information data structures.

ACK event information structures (and associated events):

MM_PLAY_ACK
(MMEV_PLAY_ACK, MMEV_PLAY_ACK_FAIL)

MM_RECORD_ACK
(MMEV_RECORD_ACK, MMEV_RECORD_ACK_FAIL)

MM_RESET_ACK
(MMEV_RESET_ACK, MMEV_RESET_ACK_FAIL)

RESULT event information structures (and associated events):

MM_DISABLE_EVENTS_RESULT
(MMEV_DISABLE_EVENTS, MMEV_DISABLE_EVENTS_FAIL)

MM_ENABLE_EVENTS_RESULT
(MMEV_ENABLE_EVENTS, MMEV_ENABLEEVENTS_FAIL)

MM_OPEN_RESULT
(MMEV_OPEN, MMEV_OPEN_FAIL)

MM_RESET_RESULT
(MMEV_RESET, MMEV_RESET_FAIL)

MM_SET_PARM_RESULT
(MMEV_SETPARM, MMEV_SETPARM_FAIL)

Event information error return codes are also returned in the MM_STOP_ACK_DETAILS unRetCode field, which is used to provide details on the MM_STOP_ACK event data associated with the MMEV_STOP_ACK and MMEV_STOP_ACK_FAIL events.

Note: If the function generates a successful initiation event (e.g., MMEV_PLAY_ACK) or a successful completion or termination event (e.g., MM_ENABLE_EVENTS), the unRetCode field returns an EMMRC_OK.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Provides the version of the structure (where "v" is the version number).

Use MM_RET_CODE_VER(v) macro to check the version of the structure.

Use MM_RET_CODE_VERSION_v macro to check version v.

unRetCode

Provides an error return code. EMMRC_OK indicates success; all other values indicate an error or failure condition. (See Section 6.3, "Dialogic® Multimedia API Event Information Error Return Codes", on page 96 for related information.) Defined values include:

- EMMRC_A_FILE_OPEN_FAILED – Audio file open failed.
- EMMRC_A_INVALID_STATE – Invalid state (audio).
- EMMRC_ALREADYSTOPPED – Device operations are already stopped.
- EMMRC_AV_INVALID_STATE – Invalid state (audio/video).
- EMMRC_FAILED – Unspecified failure.
- EMMRC_FILEREAD_FAILED – File read failed.
- EMMRC_INVALID_FILEFORMAT – Invalid file format.
- EMMRC_INVALIDARG – Invalid argument.
- EMMRC_INVALIDSTATE_ERROR – Invalid state.
- EMMRC_MEMALLOC_ERROR – Memory allocation error.
- EMMRC_MEMALLOC_POOLNOTFOUND – Memory allocation pool not found.
- EMMRC_NOT_VIDEO_FILE – Video file format is invalid.
- EMMRC_OK – Successful (no error).
- EMMRC_UNKNOWN_ERROR – Unknown error.
- EMMRC_UNSUPPORTED_MODE – Unsupported mode.
- EMMRC_V_FILE_OPEN_FAILED – Video file open failed.
- EMMRC_V_INVALID_STATE – Invalid state (video).

MM_SET_PARM

```
typedef struct tagMM_SET_PARM
{
    unsigned int    unVersion;
    eMM_PARM        eParm;
    unsigned int    unParmValue;
} MM_SET_PARM, *PMM_SET_PARM;
typedef const MM_SET_PARM* CPMM_SET_PARM;
```

Description

The MM_SET_PARM structure specifies the details of a set parameter request. This structure is used as *function input* for the **mm_SetParm( )** function.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_SET_PARM_VER(v) macro to set the version of the structure.

Use MM_SET_PARM_VERSION_v macro to set to version v.

eParm

Specifies parameter to set. Defined values include the following channel-level parameter:

- EMM_REC_IFRAME_TIMEOUT – Specifies the time to wait for an I-frame. Video recording, or multimedia (audio and video) recording, starts when an I-frame is detected or when the time-out is reached.

Note: See **mm_EnableEvents( )** and the MM_VIDEO_RECORD_STARTED structure for information on the related MMEV_VIDEO_RECORD_STARTED ***optional* intermediate (non-terminating) notification event**. See also the **mm_Record( )** function.

unParmValue

Specifies the value assigned to the parameter. Valid values are indicated for the following parameters:

- EMM_REC_IFRAME_TIMEOUT – Range: 0 - 0x7FFFFFFF. Units: ms. Default: 5000 (5 seconds). Example: Set to 9000 to specify 9 seconds. Zero (0) causes an immediate time-out and starts recording immediately.

MM_STOP

```
typedef struct tagMM_STOP
{
    unsigned int          unVersion;
    eMM_ITEM              ItemChain;
    eMM_STOP              ItemType;
    MM_STOP_DETAILS       details;
    struct tagMM_STOP*    next;
    struct tagMM_STOP*    prev;   /* optional */
} MM_STOP, *PMM_STOP;
typedef const MM_STOP* CPMM_STOP;
```

Description

The MM_STOP structure specifies the details of a stop device operations request. This structure is used as *function input* for the **mm_Stop( )** function.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_STOP_VER(v) macro to set the version of the structure.

Use MM_STOP_VERSION_v macro to set to version v.

ItemChain

Specifies the next list element for iteration. Defined values include:

- EMM_ITEM_CONT – Indicates that the next list element immediately follows the current.
- EMM_ITEM_LINK – Indicates that the next and previous list elements linked to the current via pointers.
- EMM_ITEM_EOT – Indicates that the current list element is the last in the list.

ItemType

Specifies the type of the stop element. Defined values include:

- EMM_STOP_AUDIO_PLAY – Indicates that the current item specifies stop audio play element.
- EMM_STOP_AUDIO_RECORD – Indicates that the current item specifies stop audio record element.
- EMM_STOP_VIDEO_PLAY – Indicates that the current item specifies stop video play element.
- EMM_STOP_VIDEO_RECORD – Indicates that the current item specifies stop video record element.

details

Specifies the details of the stop element. See definition of MM_STOP_DETAILS structure.

Note: Although this structure is **reserved for future use**, you must pass the structure and set the unRFU field to 0.

next
: Points to the next MM_STOP item. Needed only when ItemChain specifies a linked list element. NULL value indicates last item.

prev
: Points to the previous MM_STOP item (optional). Needed only when ItemChain specifies a linked list element. NULL value indicates first item.

MM_STOP_ACK

```
typedef struct tagMM_STOP_ACK
{
    unsigned int         unVersion;
    unsigned int         unCount;
    MM_STOP_ACK_DETAILS  details[MAX_STOP_ACK];
} MM_STOP_ACK, *PMM_STOP_ACK;
```

Description

The MM_STOP_ACK structure is an *event information* data structure is used for analyzing the results of the **mm_Stop( )** function **initiation** as reported by the MMEV_STOP_ACK or MMEV_STOP_ACK_FAIL event.

Field Descriptions

The fields of the structure are described as follows:

unVersion
: Provides the version of the structure (where "v" is the version number).

  Use MM_STOP_ACK_VER(v) macro to check the version of the structure.

  Use MM_STOP_ACK_VERSION_v macro to check version v.

unCount
: Specifies the number of MM_STOP_ACK_DETAILS elements that follow. Maximum value is 16.

details
: Specifies the details of completion. See definition of MM_STOP_ACK_DETAILS structure.

MM_STOP_ACK_DETAILS

```
typedef struct tagMM_STOP_ACK_DETAILS
{
    unsigned int         unVersion;
    eMM_STOP             ItemType;
    unsigned int         unRetCode;
} MM_STOP_ACK_DETAILS, *PMM_STOP_ACK_DETAILS;
```

Description

The STOP_ACK_DETAILS structure is a second-level *event information* data structure under the MM_STOP_ACK structure. (See Section 4.5, "Other Data Structure Levels (_DETAILS)", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion

Provides the version of the structure (where "v" is the version number).

Use MM_STOP_ACK_DETAILS_VER(v) macro to check the version of the structure.

Use MM_STOP_ACK_DETAILS_VERSION_v macro to check version v.

ItemType

Specifies the stopped item. Defined values include:

- EMM_STOP_AUDIO_PLAY – Indicates that the current item specifies stop audio play element.
- EMM_STOP_AUDIO_RECORD – Indicates that the current item specifies stop audio record element.
- EMM_STOP_VIDEO_PLAY – Indicates that the current item specifies stop video play element.
- EMM_STOP_VIDEO_RECORD – Indicates that the current item specifies stop video record element.

unRetCode

Provides an error return code for the stopped item. EMMRC_OK indicates success; all other values indicate an error or failure condition. (See Section 6.3, "Dialogic® Multimedia API Event Information Error Return Codes", on page 96 for related information.) Defined values applicable to this operation include:

- EMMRC_A_INVALID_STATE – Invalid state (audio).
- EMMRC_ALREADYSTOPPED – Device operations are already stopped.
- EMMRC_INVALIDARG – Invalid argument.
- EMMRC_V_INVALID_STATE – Invalid state (video).

MM_STOP_DETAILS

```
typedef struct tagMM_STOP_DETAILS
{
    unsigned int    unVersion;
    unsigned int    unRfu;
} MM_STOP_DETAILS, *PMM_STOP_DETAILS;
typedef const MM_STOP_DETAILS* CPMM_STOP_DETAILS;
```

Description

Note: Although this structure is **reserved for future use**, you must pass the structure and set the unRFU field to 0.

This structure is a second-level *function input* data structure under the MM_STOP structure, which is used as *function input* for the **mm_Stop( )** function. (See Section 4.5, "Other Data Structure Levels (_DETAILS)", on page 57.)

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_STOP_DETAILS_VER(v) macro to set the version of the structure.

Use MM_STOP_DETAILS_VERSION_v macro to set to version v.

unRFU

Reserved for future use. Must be set to 0.

MM_VIDEO_CODEC

```
typedef struct tagMM_VIDEO_CODEC
{
    unsigned int            unVersion;
    eMM_VIDEO_CODING        Coding;
    eMM_VIDEO_PROFILE       Profile;
    eMM_VIDEO_LEVEL         Level;
    eMM_VIDEO_IMAGE_WIDTH   ImageWidth;
    eMM_VIDEO_IMAGE_HEIGHT  ImageHeight;
    eMM_VIDEO_BITRATE       BitRate;
    eMM_VIDEO_FRAMESPERSEC  FramesPerSec;
] MM_VIDEO_CODEC, *PMM_VIDEO_CODEC;
trpedef const MM_VIDEO_CODEC* CPMM_VIDEO_CODEC
```

Description

The MM_VIDEO_CODEC structure specifies the characteristics of the video coder. This structure is a nested *function input* structure for the **mm_Play( )** or **mm_Record( )** function. It is nested directly under the MM_MEDIA_VIDEO structure, and it is a sixth-level structure under the MM_PLAY_RECORD_INFO (MM_PLAY_INFO, MM_RECORD_INFO) structure. (For a description of these *nesting levels*, see Section 4.4, "Play/Record Data Structure Levels", on page 57.)

Note: For **mm_Record( )**, none of the fields in this data structure may be modified, as video transcoding is not currently supported.

Field Descriptions

The fields of the structure are described as follows:

unVersion

Sets the version of the structure (where "v" is the version number).

Use MM_VIDEO_CODEC_VER(v) macro to set the version of the structure.

Use MM_VIDEO_CODEC_VERSION_v macro to set to version v.

Coding

Specifies the type of video coding. Defined values include:

- EMM_VIDEO_CODING_DEFAULT – (H.263)
- EMM_VIDEO_CODING_H263

Profile

Specifies the video profile. Defined values include:

- EMM_VIDEO_PROFILE_DEFAULT – (Profile 0)
- EMM_VIDEO_PROFILE_0

Level

Specifies the video signal level. Defined values include:

- EMM_VIDEO_LEVEL_DEFAULT – (Level 10)
- EMM_VIDEO_LEVEL_10
- EMM_VIDEO_LEVEL_20
- EMM_VIDEO_LEVEL_30

ImageWidth

Specifies the width of the video image in pixels. Defined values include:

- EMM_VIDEO_IMAGE_WIDTH_DEFAULT – 176 (QCIF)
- EMM_VIDEO_IMAGE_WIDTH_128 – Sub-QCIF
- EMM_VIDEO_IMAGE_WIDTH_176 – QCIF
- EMM_VIDEO_IMAGE_WIDTH_352 – CIF

ImageHeight

Specifies the height of the video image in pixels. Defined values include:

- EMM_VIDEO_IMAGE_HEIGHT_DEFAULT – 144 (QCIF)
- EMM_VIDEO_IMAGE_HEIGHT_96 – Sub-QCIF
- EMM_VIDEO_IMAGE_HEIGHT_144 – QCIF
- EMM_VIDEO_IMAGE_HEIGHT_288 – CIF

BitRate

Specifies the bitrate of the video signal. Defined values include:

- EMM_VIDEO_BITRATE_DEFAULT – 0 (zero). For **mm_Play( )**, this value sets the output bitrate to the file bitrate. For **mm_Record( )**, this value sets the file bitrate to the input bitrate. No transrating is performed.

FramesPerSec

Specifies the video frame rate. Defined values include:

- EMM_VIDEO_FRAMESPERSEC_DEFAULT – 15 fps
- EMM_VIDEO_FRAMESPERSEC_6 – 6 fps
- EMM_VIDEO_FRAMESPERSEC_15 – 15 fps
- EMM_VIDEO_FRAMESPERSEC_25 – 25 fps
- EMM_VIDEO_FRAMESPERSEC_2997 – 29.97 fps
- EMM_VIDEO_FRAMESPERSEC_30 – 30 fps

MM_VIDEO_RECORD_STARTED

```
typedef struct tagMM_VIDEO_RECORD_STARTED
{
    unsigned int    unVersion;
    unsigned int    unStatus;
} MM_VIDEO_RECORD_STARTED, *PMM_VIDEO_RECORD_STARTED;
```

Description

The MM_VIDEO_RECORD_STARTED *event information* data structure is used for analyzing the results of an **mm_Record( )** function as reported by the MMEV_VIDEO_RECORD_STARTED ***optional*** **intermediate (non-terminating) notification event** (enabled by default or by the **mm_EnableEvents( )** function).

Field Descriptions

The fields of the structure are described as follows:

unVersion

Provides the version of the structure (where "v" is the version number).

Use MM_VIDEO_RECORD_STARTED_VER(v) macro to check the version of the structure.

Use MM_VIDEO_RECORD_STARTED_VERSION_v macro to check version v.

unStatus

Specifies the status of I-frame detection. Defined values include:

- EMM_VIDEO_RCRD_IFRAME_DETECTED – I-frame detected
- EMM_VIDEO_RCRD_IFRAME_TIMEOUT – I-frame detection time-out is reached

Note: See also MM_SET_PARM for information on the EMM_REC_IFRAME_TIMEOUT parameter. See also the **mm_Record( )** function.

Multimedia API Errors 6

This chapter describes the errors supported by the Dialogic® Multimedia API and covers the following topics:

- Overview of Dialogic® Multimedia API Errors 93
- Dialogic® Multimedia API Function Error Codes 95
- Dialogic® Multimedia API Event Information Error Return Codes. 96
- Terminating and Non-Terminating Play/Record Errors. 97

6.1 Overview of Dialogic® Multimedia API Errors

The following information describes the basic categories of errors generated in the Dialogic Multimedia API:

- Function Error Codes for EMM_FAILURE
- Unsolicited Failure Event (MMEV_ERROR) Error Information
- Failure Termination Events (MMEV_xxxx_FAIL) and Error Information

Function Error Codes for EMM_FAILURE

The Multimedia API functions return a value, which in most cases, is EMM_SUCCESS (or 0) for a successful result and EMM_ERROR (or -1) for an error or an unsuccessful result.

If a Multimedia API function returns EMM_ERROR to indicate a failure, use the **mm_ErrorInfo( )** function to retrieve the reason for the error. The **mm_ErrorInfo( )** function outputs in the MM_INFO data structure an error value and error message specific to the Multimedia API function. To retrieve the correct information, the application program must call **mm_ErrorInfo( )** function immediately after the Multimedia API function fails, otherwise, the MM_INFO data may be outdated or invalid.

Note: Multimedia API function errors are thread-specific (they are only in scope for that thread).

For a list of the *function error codes* and messages returned in the MM_INFO structure, see Section 6.2, "Dialogic® Multimedia API Function Error Codes", on page 95.

Unsolicited Failure Event (MMEV_ERROR) Error Information

If the MMEV_ERROR unsolicited failure event occurs, use **mm_GetMetaEvent( )** to retrieve the reason for the error. The **mm_GetMetaEvent( )** function outputs the MM_ERROR_RESULT event data associated with the metaevent in the MM_METAEVENT data structure. To retrieve the correct information, the application must call **mm_GetMetaEvent( )** immediately after the failure event arrives and before the next Multimedia API event is requested, otherwise, the metaevent data

may be outdated or invalid. For a list of the error codes and messages returned in the MM_INFO structure, see Section 6.2, "Dialogic® Multimedia API Function Error Codes", on page 95.

Failure Termination Events (MMEV_xxxx_FAIL) and Error Information

If an error occurs during execution of an asynchronous function, a failure termination event is sent to the application. The Multimedia API failure termination events contain an MMEV_ prefix and a _FAIL suffix (MMEV_xxxx_FAIL); for example, MMEV_RECORD_FAIL.

If a Multimedia API function generates a failure termination event, use the **mm_GetMetaEvent( )** function to retrieve the reason for the error. The **mm_GetMetaEvent( )** function outputs metaevent information in the MM_METAEVENT data structure. If the MMME_MM_EVENT bit is set in the flags field, it indicates that it is a Multimedia API event, and the evtdatap field points to event data that contains failure information specific to the Multimedia API function. To retrieve the correct information, the application must call **mm_GetMetaEvent( )** immediately after the failure event arrives and before the next Multimedia API event occurs, otherwise, the metaevent data may be outdated or invalid.

There are two error information types for failure termination events:

MMEV_PLAY_FAIL and MMEV_RECORD_FAIL Termination Event Error Information
: If an **mm_Play( )** or **mm_Record( )** function generates a MMEV_PLAY_FAIL or MMEV_RECORD_FAIL *failure termination event*, see Section 6.4, "Terminating and Non-Terminating Play/Record Errors", on page 97 for information on how to retrieve the error information.

Event Information Error Return Codes (EMMRC_xxxx)
: The metaevent information that is associated with a *non-media I/O operation failure termination event* contains an error return code. These *event information error return codes* apply to any metaevent (provided by **mm_GetMetaEvent( )**) for which the event data is of type MM_RET_CODE, which includes the following event information data structures:

  ACK event information structures

  - MM_PLAY_ACK
  - MM_RECORD_ACK
  - MM_RESET_ACK

  RESULT event information structures

  - MM_DISABLE_EVENTS_RESULT
  - MM_ENABLE_EVENTS_RESULT
  - MM_OPEN_RESULT
  - MM_RESET_RESULT

- MM_SET_PARM_RESULT

The MM_RET_CODE event information structure provides in its unRetCode field the error return code related to an event.

Event information error return codes are also returned in the MM_STOP_ACK_DETAILS unRetCode field, which is used to provide details on the MM_STOP_ACK event data associated with the MMEV_STOP_ACK and MMEV_STOP_ACK_FAIL events.

The unRetCode field returns an error return code (see Section 6.3, "Dialogic® Multimedia API Event Information Error Return Codes", on page 96).

Note: If the function generates a successful initiation event (e.g., MMEV_PLAY_ACK) or a successful completion or termination event (e.g., MM_ENABLE_EVENTS), the unRetCode field returns an EMMRC_OK.

6.2 Dialogic® Multimedia API Function Error Codes

The function error codes are defined in *mmerrs.h* header file. The API can generate the following function error codes (listed in alphabetical order):

EMM_BADDEV
: Invalid device descriptor.

EMM_BADPARM
: Invalid parameter in function call.

EMM_BADPROD
: Function is not supported on this board.

EMM_BUSY
: Device is already busy.

EMM_FWERROR
: Firmware error.

EMM_IDLE
: Device is idle.

EMM_NOERROR
: No errors.

EMM_NOSUPPORT
: Data format not supported.

EMM_NOTIMP
: Function is not implemented.

EMM_SYSTEM
: System error.

EMM_TIMEOUT
: Function timed out.

6.3 Dialogic® Multimedia API Event Information Error Return Codes

The event information error return codes are defined in the *mmerrs.h* header file. The API can generate the following event information error return codes (listed in alphabetical order):

EMMRC_A_FILE_OPEN_FAILED
Audio file open failed.

EMMRC_A_INVALID_STATE
Invalid state (audio).

EMMRC_ALREADYSTOPPED
Device operations are already stopped.

EMMRC_AV_INVALID_STATE
Invalid state (audio/video).

EMMRC_FAILED
Unspecified failure.

EMMRC_FILEREAD_FAILED
File read failed.

EMMRC_INVALIDARG
Invalid argument.

EMMRC_INVALID_FILEFORMAT
Invalid file format.

EMMRC_INVALIDSTATE_ERROR
Invalid state.

EMMRC_MEMALLOC_ERROR
Memory allocation error.

EMMRC_MEMALLOC_POOLNOTFOUND
Memory allocation pool not found.

EMMRC_NOT_VIDEO_FILE
Video file format is invalid.

EMMRC_OK
Successful (no error).

EMMRC_UNKNOWN_ERROR
Unknown error.

EMMRC_UNSUPPORTED_MODE
Unsupported mode.

EMMRC_V_FILE_OPEN_FAILED
Video file open failed.

EMMRC_V_INVALID_STATE
Invalid state (video).

6.4 Terminating and Non-Terminating Play/Record Errors

The **mm_Play( )** function generates an MMEV_PLAY_FAIL termination event upon encountering an error during the playback operation. The **mm_Record( )** function generates an MMEV_RECORD_FAIL termination event upon completion or termination of the record operation. It indicates that an *intermediate (non-terminating) error* occurred during the operation. The following information describes the error information for these failure events, as well as the corresponding non-failure events:

MMEV_PLAY_FAIL and MMEV_PLAY

The metaevent data associated with these termination events is provided in an MM_PLAY_CMPLT (MM_PLAY_RECORD_CMPLT) structure. Within this structure, details on the event termination are provided by the MM_PLAY_RECORD_CMPLT_DETAILS details second-level data structure.

Upon encountering an error, the playback terminates with an **MMEV_PLAY_FAIL**. The Reason field in the details structure indicates an EMM_TR_ERROR, and the Status field in the details structure indicates the type of error:

- EMM_STATUS_PLAY_A_FILEREAD_ERROR – Audio file read error.
- EMM_STATUS_PLAY_V_FILEREAD_ERROR – Video file read error.
- EMM_STATUS_PLAY_V_ERROR_FS_GT_MFS – Video frame size greater than maximum frame size of internal buffer.

If the playback terminates with an **MMEV_PLAY**, the Reason field in the details structure indicates a termination reason, such as MM_TR_EOF, meaning it is a successful completion (end of file), or EMM_TR_USERSTOP, which means successful termination by the **mm_Stop( )** function. The Status field in the details structure indicates:

- EMM_STATUS_SUCCESS – Successful completion or successful termination

MMEV_RECORD_FAIL and MMEV_RECORD

The metaevent data associated with these termination events is provided in an MM_RECORD_CMPLT (MM_PLAY_RECORD_CMPLT) structure. Within this structure, details on the event termination are provided by the MM_PLAY_RECORD_CMPLT_DETAILS details second-level data structure.

If the recording terminates with an **MMEV_RECORD_FAIL**, it indicates that an intermediate (non-terminating) error occurred during the record operation. The Reason field in the details structure indicates an EMM_TR_ERROR, and the Status field in the details structure indicates the type of **non-terminating error**:

- EMM_STATUS_RCRD_V_DRPD_FRAME_FULL_ERROR – Full video frame dropped error.
- EMM_STATUS_RCRD_V_PKTS_DROPD_FS_GT_MFS – Video packets dropped frame size greater than maximum frame size of internal buffer.
- EMM_STATUS_RCRD_A_DRPD_FRAME_FULL_ERROR – Full audio frame dropped error.

If the recording terminates with an **MMEV_RECORD**, the Reason field in the details structure indicates a termination reason, such as EMM_TR_USERSTOP, which indicates successful termination by the **mm_Stop( )** function. The Status field in the details structure indicates:

- EMM_STATUS_SUCCESS – Successful completion or successful termination (no error).