Dialogic®

Dialogic® IP Media Library API

Library Reference

August 2007

05-2257-010

Publication Date: August 2007

Document Number: 05-2257-010

Contents

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2257-010	August 2007	Made global changes to reflect Dialogic brand. As part of rebranding, renamed document from "IP Media Library API for HMP Library Reference" to "Dialogic® IP Media Library API Library Reference". IPM_AUDIO_CODER_INFO data structure: Updated to include CODER_TYPE_EVRC; removed CODER_TYPE_EVRC_4_8k, CODER_TYPE_EVRC_4_8k_NATIVE, CODER_TYPE_EVRC_9_6k, CODER_TYPE_EVRC_4_8k_NATIVE. Also removed CODER_TYPE_QCELP_8K_NATIVE and CODER_TYPE_QCELP_13K_NATIVE. Updated table of supported audio coder properties for AMR, EVRC, QCELP. IPM_AUDIO_CODER_OPTIONS_INFO data structure: Updated unCoderOptions (added value for QCELP); updated unParm1 and nValue1 (removed CODER_OPT_CDMA_MAX_RATE for EVRC and QCELP); updated unParm2 and unValue2; updated example code.
05-2257-009	May 2007	Function Summary by Category chapter: Added ipm_InitResponseSend( ) and ipm_InitSend( ) to the Media Session Functions category. Added Dialogic® IP Media Library API Function Support by Platform section. Function Information chapter: Added the following new functions for Nb UP support: ipm_InitResponseSend( ) and ipm_InitSend( ). ipm_GetCapabilities( ) function: Corrected example code. ipm_GetLocalMediaInfo( ) function: Added statement about querying RTP/RTCP information. Added caution for Dialogic® Multimedia Platform for AdvancedTCA (Multimedia Platform for AdvancedTCA). ipm_SetRemoteMediaInfo( ) function: Removed from this document. Deprecated function. Replaced with ipm_StartMedia( ). ipm_StartMedia( ) function: Added caution for Multimedia Platform for AdvancedTCA. Updated example code to illustrate video. Events chapter: Added the following events: IPMEV_INIT_COMPLETE, IPMEV_INIT_FAILED, IPMEV_INIT_RESPONSE_SEND, IPMEV_INIT_SEND, IPMEV_INIT_RECEIVED, IPMEV_NACK_SENT. Data Structures chapter: Added the following new structures for Nb UP support: IPM_INIT_FAILURE, IPM_INIT_RECEIVED, IPM_INIT_RESPONSE, IPM_INIT_SEND, IPM_NACK_SENT, IPM_NBUP_INIT_FAILURE, IPM_NBUP_INIT_RECEIVED, IPM_NBUP_INIT_RESPONSE, IPM_NBUP_INIT_SEND, IPM_NBUP_NACK_SENT, IPM_NBUP_PROFILE_INFO, IPM_NBUP_RFCI_INFO, and IPM_NBUP_SUBFLOW_INFO. CT_DEVINFO data structure: Added CT_DFHMPATCA value for ct_devfamily. IPM_AUDIO_CODER_INFO data structure: Added several new coder types, including AMR-NB, EVRC, QCELP, and all _NATIVE coder types. Updated table of supported audio coder properties. IPM_AUDIO_CODER_OPTIONS_INFO data structure: New.

Document No.	Publication Date	Description of Revisions
05-2257-009 (cont.)	May 2007	IPM_MEDIA data structure: Added AudioCoderOptionsInfo field. Added eMediatype values, including MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO and MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO. Corrected MEDIATYPE_FAX_SIGNAL value (previously listed as MEDIATYPE_FAX_SIGNAL_INFO). IPM_PARM_INFO data structure: Added eIP_PARM defines, including PARMCH_LATENCYALG_AUDIO, PARMCH_LATENCYFRAMEINIT_AUDIO, and PARMCH_LATENCYFRAMEMAX_AUDIO. IPM_VIDEO_CODER_INFO data structure: Removed unProfileLevelID and szVisualConfiguration fields because they are not yet supported. Error Codes chapter: Updated for ipm_InitResponseSend( ) and ipm_InitSend( ).
05-2257-008	August 2006	ipm_GetCapabilities( ) function: Added.
05-2257-007	June 2006	Media Session Functions section: Added ipm_SecurityGenMasterKeys( ) function. ipm_GetLocalMediaInfo( ) function: Updated description to clarify eMediaType and unCount as members of data structures referenced by the pMediaInfo function parameter and enumerate the allowed values. Also udated for video support. ipm_SecurityGenMasterKeys( ) function: Added for SRTP support. Events chapter: Updated the IPMEV_QOS_ALARM event description. Events chapter: Added IPMEV_SEC_NOTIFY_EXPIRE_KEY_AUDIO and IPMEV_SEC_NOTIFY_EXPIRE_KEY_VIDEO event descriptions. Data Structures chapter: Added SRTP and video related structures. IPM_AUDIO_CODER_INFO data structure: Added to replace IPM_CODER_INFO. IPM_PARM_INFO data structure: In the PARMCH_ECNLP_ACTIVE parameter description in the table, updated the definition of “NLP”. IPM_QOS_ALARM_DATA data structure: Updated to include SRTP and SRTCP alarms. IPM_QOS_THRESHOLD_DATA data structure: Updated to include SRTP and SRTCP alarms. IPM_MEDIA data structure: Updated for SRTP and video support. IPM_SECURITY_BASE64_KEY data structure: Added for SRTP support. IPM_SECURITY_BINARY_KEY data structure: Added for SRTP support. IPM_SECURITY_INFO data structure: Added for SRTP support. IPM_SECURITY_KEY data structure: Added for SRTP support. IPM_SRTP_PARMS data structure: Added for SRTP support. IPM_VIDEO_CODER_INFO data structure: Added for video support

Document No.	Publication Date	Description of Revisions
05-2257-006	December 2005	Media Session Functions section: Added ipm_ModifyMedia( ) function Events section: Added IPMEV_MODIFY_MEDIA and IPMEV_MODIFY_MEDIA_FAIL ipm_GetLocalMediaInfo( ) function: Removed references to video ipm_ModifyMedia( ) function: Added to support endpoint confiugration for half- and full-duplex media streams ipm_SetRemoteMediaInfo( ): Removed detailed information since the function is deprecated ipm_StartMedia( ) function: Added the DATA_IP_INACTIVE option to the eDirection parameter. Rephrased the first caution for greater clarity. IPM_AUDIO_CODER_INFO data structure: Removed from this version IPM_CODER_INFO data structure: Updated the supported coders table; removed the original note and added a note about G.726 usage. IPM_MEDIA data structure: Removed references to video; updated names of elements for audio coders IPM_PARM_INFO data structure: Added the following parameters: PARMBD_RTCPAUDIO_INTERVAL PARMBD_RTPAUDIO_PORT_BASE PARMCH_ECACTIVE PARMCH_ECHOTAIL PARMCH_ECNLP_ACTIVE IPM_VIDEO_CODER_INFO data structure: Removed from this verson General: Hid all references to deprecated ipm_SetRemoteMediaInfo( ) function except the function reference page itself.
05-2257-005	August 2005	ipm_GetLocalMediaInfo( ) function: Updated code example for video media info IPM_AUDIO_CODER_INFO data a structure: Added note about name change; added note to unCoderPayloadType description (PTR#33921) IPM_MEDIA data structure: Added union element for video coders; changed name of element for audio coders IPM_PARM_INFO data structure: Added info on using PARMCH_TOS for DSCP field IPM_RFC2833_SIGNALID_INFO data structure: Restored to document (previous removal was in error) IPM_VIDEO_CODER_INFO: New data structure
05-2257-004	April 2005	Function Summary by Category chapter: Removed unimplemented ipm_SendRFC2388SignalIDToIP function ipm_DisableEvents( ) function page: Removed unsupported EVT_FAXTONE and EVT_T38CALLSTATE event types ipm_EnableEvents( ) function page: Removed unsupported EVT_FAXTONE and EVT_T38CALLSTATE event types ipm_SendRFC2388SignalIDToIP function: Removed as unimplemented ipm_SetQoSThreshold( ) function page: Added Caution on possible failure scenario. Corrected code example Events chapter: Removed unimplemented IPMEV_RFC2833SIGNALRECEIVED event IPM_RFC2833_SIGNALID_INFO data structure section: Removed as unused IPM_QOS_SESSION_INFO data structure page: Clarified descriptions of jitter and lost packets QoS statistics

Document No.	Publication Date	Description of Revisions
05-2257-003	September 2004	ipm_GetQoSAlarmStatus( ) function: Corrected code example ipm_GetSessionInfo( ) function: Added info on NULL pointer in asynch mode ipm_GetXmitSlot( ) function: Added info on NULL pointer in asynch mode ipm_Open( ) function: Removed caution that function must be called in synchronous mode ipm_ReceiveDigits( ) function: Clarified usage of data structure. Added caution regarding active RTP session requirement for receiving digits. ipm_SendDigits( ) function: Added info about maximum number of digits. Added caution regarding active RTP session requirement for sending digits. ipm_StartMedia( ) function: Added caution to avoid setting IP address 0.0.0.0 (PTR#32986). Corrected name of completion event. CT_DEVINFO data structure: Removed much information irrelevant to the structure's use with IPML on HMP IPM_DIGIT_INFO data structure: Added info about maximum number of digits and send vs. receive usage differences IPM_PARM_INFO data structure: Removed descriptions of three unsupported RFC2833 parameters IPM_PORT_INFO data structure: Added caution to avoid setting IP address 0.0.0.0 (PTR#32986)
05-2257-002	April 2004	Made document HMP-specific by removing hardware-specific information, including "IP Media Function Support by Platform" section. ipm_DisableEvents( ) function: Removed two unsupported events. Added EVT_RTCPTIMEOUT and EVT_RTPTIMEOUT QoS alarm events. ipm_EnableEvents( ) function: Removed two unsupported events. Added EVT_RTCPTIMEOUT and EVT_RTPTIMEOUT QoS alarm events. ipm_GetLocalMediaInfo( ) function: Corrected MEDIATYPE... types referred to in the Cautions and Code Example sections ipm_Ping( ): Removed as unsupported ipm_SendDigits( ) function: Documented as supported for HMP ipm_SetQoSThreshold( ) function: Revised code example to use correct minimum value (100) for unTimeInterval ipm_SetRemoteMediaInfo( ) function: Documented function as deprecated. Corrected code example. ipm_StartMedia( ) function: Corrected code example ipm_Stop( ) function: Removed two unsupported eIPM_STOP_OPERATION values IPM_CODER_INFO data structure: Removed hardware-specific enum values and tables of supported coders. Updated table of supported coders. IPM_PARM_INFO data structure: Added PARMCH_RX_ADJVOLUME and PARMCH_RX_ADJVOLUME parameters. Expanded and corrected descriptions of all parameters. Removed hardware-specific enum values. IPM_PING_INFO and IPM_PINGPARM structures: Removed as unsupported IPM_QOS_ALARM_DATA data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms. IPM_QOS_SESSION_INFO data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms.

Document No.	Publication Date	Description of Revisions
05-2257-002 (cont.)	April 2004	IPM_QOS_THRESHOLD_DATA data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms. Added value ranges for unFaultThreshold. Updated descriptions of all fields. Added table of default parameter values. Error Codes chapter: added EIPM_RESOURCEINUSE
05-2257-001	September 2003	Initial version of HMP-specific document. Much of the information contained in this document was previously published in the *IP Media Library API for Linux and Windows Operating Systems Library Reference*, document number 05-1833-002. Among other small changes, the following changes were made: ipm_GetCTInfo( ): New function CT_DEVINFO: Added to book IPM_PARM_INFO data structure: Added valid value and variable type info for PARMCH_RFC2833EVT_TX_PLT and PARMCH_RFC2833EVT_RX_PLT. Deleted PARMCH_RFC2833MUTE_AUDIO, PARMCH_RFC2833TONE_TX_PLT, and PARMCH_RFC2833TONE_RX_PLT

About This Publication

The following topics provide information about this publication:

Purpose

This guide provides details about the Dialogic® IP Media Library API, including function descriptions, event messages, data structures, and error codes. This is a companion guide to the *Dialogic® IP Media Library API Programming Guide*, which provides information on developing applications using the Dialogic® IP Media Library API.

This guide specifically applies to the Dialogic® IP Media Library API that is provided with the Dialogic® Host Media Processing (HMP) Software and the Dialogic® Multimedia Platform for AdvancedTCA software. The Dialogic® IP Media Library API as implemented in the Dialogic® System Release software has some significant functional differences from the Dialogic® HMP Software implementation, and separate versions of the IP Media Library documents are provided with System Release software releases.

Applicability

This document version (05-2257-010) is published for Dialogic® Host Media Processing Software Release 3.1LIN.

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

Intended Audience

This guide is intended for software developers who will access the Dialogic® IP Media Library API. This may include any of the following:

- Distributors
- System Integrators

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

How to Use This Publication

Refer to this publication after you have installed the hardware and the system software which includes the Dialogic® IP Media Library API. This publication assumes that you are familiar with the Linux or Windows® operating system and the C programming language.

The information in this guide is organized as follows:

- Chapter 1, "Function Summary by Category" groups the IP media APIs into categories.
- Chapter 2, "Function Information" provides details about each IP media API function, including parameters, return values, events, and error codes.
- Chapter 3, "Events" describes the events returned by the Dialogic® IP Media Library API.
- Chapter 4, "Data Structures" provides details about each data structure used by the Dialogic® IP Media Library API, including fields and descriptions.
- Chapter 5, "Error Codes" lists the error codes included in the Dialogic® IP Media Library API.

Related Information

For related Dialogic publications, see the product documentation (known as the online bookshelf) provided with the software release or at the following web site:
http://www.dialogic.com/manuals/default.htm

Function Summary by Category 1

The Dialogic® IP Media Library API contains functions which control and monitor media resources in an IP environment. This chapter contains an overview of the Dialogic® IP Media Library API functions, which are grouped into the categories listed below. This chapter also includes a table listing function support on various platforms.

1.1 System Control Functions

The following functions are used to manage channel, parameter, and event operations:

ipm_Close( )
closes an IP channel

ipm_DisableEvents( )
disables IP notification events

ipm_EnableEvents( )
enables IP notification events

ipm_GetParm( )
returns IP channel parameters

ipm_GetXmitSlot( )
returns TDM time slot information for an IP channel

ipm_Listen( )
connects an IP channel to a TDM time slot

ipm_Open( )
opens an IP channel and returns a handle

ipm_SetParm( )
sets IP channel parameters

ipm_UnListen( )
disconnects an IP channel from a TDM time slot

1.2 I/O (Input/Output) Functions

The following functions are used to transfer digits and data:

ipm_ReceiveDigits( )
enables the IP channel to receive digits from the TDM bus

ipm_SendDigits( )
generates supplied digits to the TDM bus

1.3 Media Session Functions

The following functions are used to perform session management:

ipm_GetCTInfo( )
retrieves information about an IPM device voice channel

ipm_GetLocalMediaInfo( )
retrieves properties for the local media channel

ipm_GetSessionInfo( )
retrieves statistics for the current session

ipm_InitResponseSend( )
sends a response to an IP session

ipm_InitSend( )
sends an initialization message to a remote party

ipm_ModifyMedia( )
modifies the properties of an active media session

ipm_SecurityGenMasterKeys( )
generate master and salt keys

ipm_StartMedia( )
sets properties for the local and remote media channels and starts the session

ipm_Stop( )
stops operations on an IP channel

1.4 Quality of Service (QoS) Functions

The following functions are used to control QoS alarms and alarm thresholds:

ipm_GetQoSAlarmStatus( )
retrieves the ON/OFF state of QoS alarms

ipm_GetQoSThreshold( )
retrieves QoS alarm threshold settings

ipm_ResetQoSAlarmStatus( )
resets QoS alarm to OFF state once it has been triggered

ipm_SetQoSThreshold( )
changes QoS alarm threshold settings

1.5 Dialogic® IP Media Library API Function Support by Platform

The following table provides an alphabetical listing of Dialogic® IP Media Library API functions. The table indicates which platforms are supported for each of the functions: Dialogic® Host Media Processing (HMP) Software or Dialogic® Multimedia Platform for AdvancedTCA.

Although a function may be supported on all platforms, there may be some differences on its use. For details, see the function reference descriptions in Chapter 2, "Function Information".

Table 1. Dialogic® IP Media Library API Function Support by Platform

Function	Dialogic® HMP Software	Dialogic® Multimedia Platform for AdvancedTCA
ipm_Close( )	S	S
ipm_DisableEvents( )	S	S
ipm_EnableEvents( )	S	S
ipm_GetCapabilities( )	S	S
ipm_GetCTInfo( )	S	S
ipm_GetLocalMediaInfo( ) †	S	S
ipm_GetParm( ) †	S	S
ipm_GetQoSAlarmStatus( )	S	S
ipm_GetQoSThreshold( )	S	S
ipm_GetSessionInfo( )	S	S
ipm_GetXmitSlot( )	S	S
ipm_InitResponseSend( )	S	S
ipm_InitSend( )	S	S
ipm_Listen( )	S	S
ipm_ModifyMedia( )	S	NS
ipm_Open( )	S	S
ipm_ReceiveDigits( )	S	NS
ipm_ResetQoSAlarmStatus( )	S	NS
ipm_SecurityGenMasterKeys( )	S	NS
ipm_SendDigits( )	S	NS

Legend:
NS = Not Supported, S = Supported,
† = Variance between platforms, refer to Function Description for more information.

Table 1. Dialogic® IP Media Library API Function Support by Platform (Continued)

Function	Dialogic® HMP Software	Dialogic® Multimedia Platform for AdvancedTCA
ipm_SetParm( ) †	S	S
ipm_SetQoSThreshold( )	S	S
ipm_SetRemoteMediaInfo( ) (use **ipm_StartMedia( )**)	deprecated	deprecated
ipm_StartMedia( ) †	S	S
ipm_Stop( )	S	S
ipm_UnListen( )	S	S
Legend: NS = Not Supported, S = Supported, † = Variance between platforms, refer to Function Description for more information.		

Function Information 2

This chapter contains a detailed description of each Dialogic® IP Media Library API function, presented in alphabetical order.

2.1 Function Syntax Conventions

Note: The "Platform" line in the function header table of each function indicates the platforms supported.

The Dialogic® IP Media Library API functions use the following format:

```
ipm_Function (DeviceHandle, Parameter1, Parameter2, …, ParameterN, Mode)
```

where:

ipm_Function
is the name of the function

DeviceHandle
is an input field that directs the function to a specific line device

Parameter1, Parameter2, …, ParameterN
are input or output fields

Mode
is an input field indicating how the function is executed. This field is applicable to certain functions only. For example, **ipm_Close( )** can only be called synchronously, so Mode is not used. Possible Mode values are:

- EV_ASYNC for asynchronous mode execution. When running asynchronously, the function will return 0 to indicate it has initiated successfully, and will generate a termination event to indicate completion.
- EV_SYNC for synchronous mode execution. When running synchronously, the function will return a 0 to indicate that it has completed successfully.

ipm_Close( )

Name: int ipm_Close(nDeviceHandle, *pCloseInfo)

Inputs: int nDeviceHandle • IP Media device handle

IPM_CLOSE_INFO *pCloseInfo • set to NULL

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: synchronous only

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_Close( )** function closes an IP channel device and disables the generation of all events.

Parameter	Description
nDeviceHandle	IP Media device handle returned by **ipm_Open( )**
pCloseInfo	set to NULL; reserved for future use

Termination Events

None - this function operates in synchronous mode only.

Cautions

- The **pCloseInfo** pointer is reserved for future use and must be set to NULL.
- Issuing a call to **ipm_Open( )** or **ipm_Close( )** while the device is being used by another process will not affect the current operation of the device. Other handles for that device that exist in the same process or other processes will still be valid. The only process affected by **ipm_Close( )** is the process that called the function.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_CONFIG
Configuration error

EIPM_FWERROR
Firmware error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void main()
{
    int nDeviceHandle;

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Application is shutting down.
    Need to close IP device handle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_Close(nDeviceHandle, NULL) == -1)
    {
        printf("----------->ipm_Close() failed for handle = %d\n", nDeviceHandle);
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    Continue cleanup
    .
    .
    */
}
```

See Also

- **ipm_Open( )**

ipm_DisableEvents( )

Name: int ipm_DisableEvents(nDeviceHandle, *pEvents, unNumOfEvents, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
eIPM_EVENT *pEvents	• array of events to disable
unsigned int unNumOfEvents	• number of events to disable
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

■ Description

The **ipm_DisableEvents( )** function disables IP notification events. Some events are used for Quality of Service (QoS) alarm notifications. Other events are used to indicate status, for example, if an RFC2833 event has been detected.

Notification events are different from asynchronous function termination events, such as IPMEV_OPEN, which cannot be disabled. Once a particular notification event is successfully disabled, the application is not notified if an event of that type occurs.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pEvents	array of enumerations that specifies the events to disable The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_LOSTPACKETS – QoS alarm for excessive percentage of lost packets • EVT_JITTER – QoS alarm for excessive average jitter • EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity • EVT_RTPTIMEOUT – QoS alarm for RTP inactivity • EVT_RFC2833 – RFC 2833 event • EVT_SEC_AUTH_FAIL_AUDIO – Secure RTP QoS alarm for authentication failure on audio packets • EVT_SEC_AUTH_FAIL_VIDEO – Secure RTP QoS alarm for authentication failure on video packets • EVT_SEC_MKI_NOMATCH_AUDIO – Secure RTP QoS alarm for MKI mis-match on audio packets • EVT_SEC_MKI_NOMATCH_VIDEO – Secure RTP QoS alarm for MKI mis-match on video packets • EVT_SEC_NOTIFY_EXPIRE_KEY_AUDIO – Secure RTP notification of encryption key for audio about to expire • EVT_SEC_NOTIFY_EXPIRE_KEY_VIDEO – Secure RTP notification of encryption key for video about to expire • EVT_SEC_PKT_REPLAY_AUDIO – Secure RTP QoS alarm for replay detection on audio packets • EVT_SEC_PKT_REPLAY_VIDEO – Secure RTP QoS alarm for replay detection on video packets
unNumOfEvents	number of events to disable (number of enumerations in **pEvents** array)
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_EVENT_DISABLED
: Indicates successful completion; that is, specified events were disabled. This event does not return any data.

IPMEV_ERROR
: Indicates that the function failed.

Cautions

The function fails if **nDeviceHandle** specifies a board device; notification events are only supported for channel devices.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_EVT
Invalid event

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

EIPM_UNSUPPORTED
Function unsupported

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    eIPM_EVENT myEvents[2] ={EVT_LOSTPACKETS, EVT_JITTER};
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV, EV_ANYEVT, (HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Application is shutting down
    Need to disable all enabled events for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open() and
    The events listed in myEvents were enabled sometime earlier.
    */
    if(ipm_DisableEvents(nDeviceHandle, myEvents, 2, EV_ASYNC) == -1)
    {
        printf("ipm_DisableEvents failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
```

```
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    Continue shut down
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_DisableEvents */
        case IPMEV_EVENT_DISABLED:
            printf("Received IPMEV_EVENT_DISABLED for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_EnableEvents( )**

ipm_EnableEvents( )

Name: int ipm_EnableEvents(nDeviceHandle, *pEvents, unNumOfEvents, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
eIPM_EVENT *pEvents	• array of events to enable
unsigned int unNumOfEvents	• number of events to enable
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

■ Description

The **ipm_EnableEvents( )** function enables IP notification events. Some events are used for Quality of Service (QoS) notifications on a particular media channel. Other events are used to indicate status, for example, if an RFC2833 event has been detected.

Notification events (solicited events) are different from asynchronous function termination events, such as IPMEV_OPEN, which cannot be disabled. Once a particular notification event is successfully enabled, the application is notified via SRL event management functions whenever the specified event occurs.

Note: A SUCCESS message returned from a set event request in the IPML library is only a notification that the request was processed, not that all requested events are enabled.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pEvents	array of enumerations that specifies the events to enable The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_LOSTPACKETS – QoS alarm for excessive percentage of lost packets • EVT_JITTER – QoS alarm for excessive average jitter • EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity • EVT_RTPTIMEOUT – QoS alarm for RTP inactivity • EVT_RFC2833 – RFC 2833 event • EVT_SEC_AUTH_FAIL_AUDIO – Secure RTP QoS alarm for authentication failure on audio packets • EVT_SEC_AUTH_FAIL_VIDEO – Secure RTP QoS alarm for authentication failure on video packets • EVT_SEC_MKI_NOMATCH_AUDIO – Secure RTP QoS alarm for MKI mis-match on audio packets • EVT_SEC_MKI_NOMATCH_VIDEO – Secure RTP QoS alarm for MKI mis-match on video packets • EVT_SEC_NOTIFY_EXPIRE_KEY_AUDIO – Secure RTP notification of encryption key for audio about to expire • EVT_SEC_NOTIFY_EXPIRE_KEY_VIDEO – Secure RTP notification of encryption key for video about to expire • EVT_SEC_PKT_REPLAY_AUDIO – Secure RTP QoS alarm for replay detection on audio packets • EVT_SEC_PKT_REPLAY_VIDEO – Secure RTP QoS alarm for replay detection on video packets
unNumOfEvents	number of events to enable (number of enumerations in **pEvents** array)
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_EVENT_ENABLED
: Indicates successful completion; that is, specified events were enabled. This event does not return any data.

IPMEV_ERROR
: Indicates that the function failed.

Cautions

The function fails if **nDeviceHandle** specifies a board device; notification events are only supported for channel devices.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_EVT_EXIST
Event already enabled

EIPM_EVT_LIST_FULL
Too many events

EIPM_INTERNAL
Internal error

EIPM_INV_EVT
Invalid event

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

EIPM_UNSUPPORTED
Function unsupported

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    eIPM_EVENT myEvents[2] ={EVT_LOSTPACKETS, EVT_JITTER};
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Need to enable three events for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_EnableEvents(nDeviceHandle, myEvents, 2, EV_ASYNC) == -1)
    {
        printf("ipm_EnableEvents failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
```

```
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    . Continue Processing
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        . List of expected events
        .
        .
        */

        /* Expected reply to ipm_EnableEvents() */
        case IPMEV_EVENT_ENABLED:
            printf("Received IPMEV_EVENT_ENABLED for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_DisableEvents( )**

ipm_GetCapabilities( )

Name: ipm_GetCapabilities(a_nDeviceHandle, a_CapType, a_num, a_CapabilitiesArray[], a_usMode);

Inputs:

int a_nDeviceHandle	• IP Media device handle
eCAPABILITY_TYPE a_CapType	• capability type to be retrieved
unsigned int a_num	• number of entries in the capability array
IPM_CAPABILITIES a_CapabilitiesArray[]	• capability array
unsigned short a_usMode	• mode of operation

Returns: number of capabilities available
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetCapabilities( )** function returns the number of capabilities of the specified type (for example, coders) and details of each capability supported by an active Dialogic® HMP Software license. The number of capabilities available may be greater than the number specified by the **a_num** input parameter, therefore the following rules apply:

- If **a_num** is zero and/or **a_CapabilitiesArray[]** is NULL, this function returns only the number of capabilities available; no capability detail is retrieved.
- If **a_num** is larger than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of all capabilities and the remaining allocated memory is unused.
- If **a_num** is smaller than the number of capabilities available (the return value), **a_CapabilitiesArray[]** is filled with details of **a_num** capabilities (that is, as many as will fit); details of the remaining capabilities are not retrieved.

Parameter	Description
n_DeviceHandle	handle of the IP Media device
a_CapType	capability type, for example CAPABILITY_CODERLIST
a_num	the number of entries in the capability array
a_CapabilitiesArray[]	the capability array
a_usMode	operation mode Set to EV_SYNC for synchronous execution

The datatype for the **a_CapabilitiesArray[]** parameter is a union, IPM_CAPABILITIES, which is defined as follows:

```
typedef struct ipm_capabilities_tag
{
   unsigned int version;
   union
   {
      IPM_CODER_INFO Coder;
      // Future types here.
   };
}IPM_CAPABILITIES;
```

In this union, the IPM_CODER_INFO (now known as IPM_AUDIO_CODER_INFO) data structure provides coder details such as coder type, frame size, number of frames per packet, VAD enable/disable information and payload-related information.

The datatype for the **a_CapType** parameter is eCAPABILITY_TYPE, an enumeration that is defined as follows:

```
enum eCAPABILTIY_TYPE
{
   CAPABILITY_CODERLIST;
}
```

The **ipm_GetCapabilities( )** function is supported in synchronous mode only. If asynchronous mode (a_usMode = EV_ASYNC) is specified, an error is generated.

Cautions

None.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** or **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_ERROR
System error

Example

In this example, the first **ipm_GetCapabilities**( ) call retrieves only number of capabilities available (count). That number is then used to allocate the right amount of memory and retrieve details of all the capabilities.

```
#include <ipmlib.h>

unsigned int count;
IPM_CAPABILITIES *caps;
int i;

count=ipm_GetCapabilities(dev,CAPABILITY_CODERLIST,0,NULL,EV_SYNC);
caps=(IPM_CAPABILITIES *)malloc(sizeof(IPM_CAPABILITIES)*count);

// check for memory error here

count=ipm_GetCapabilities(dev,CAPABILITY_CODERLIST,count,caps,EV_SYNC);

for (i=0;i<count;i++)
{
    printf("RFC 1890 Coder Type %ui supported\n",caps[i].Coder.unCoderPayloadType);
}

// Free coder list here

free(caps);
```

See Also

None.

ipm_GetCTInfo( )

Name: int ipm_GetCTInfo(nDeviceHandle, *pCTInfo, usMode)

Inputs:

int nDeviceHandle	• valid channel device handle
CT_DEVINFO *pCTInfo	• pointer to device information structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: ipmlib.h

Category: Media Session

Mode: Synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetCTInfo( )** function returns information about a voice channel of an IPM device. This information is contained in a CT_DEVINFO data structure.

Parameter	Description
nDeviceHandle	specifies the valid IP channel handle obtained when the channel was opened using **ipm_Open( )**
pCTInfo	specifies a pointer to the CT_DEVINFO structure that contains the IP channel device information
usMode	operation mode Set to EV_SYNC for synchronous execution. Asynchronous mode is not supported.

Cautions

This function fails if an invalid IP channel handle is specified.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return the following error:

EIPM_BADPARM
Invalid parameter

Example

```
#include <srllib.h>
#include <ipmlib.h>
#include <errno.h>
```

```
main()
{
int chdev; /* Channel device handle */
CT_DEVINFO ct_devinfo; /* Device information structure */
/* Open board 1 channel 1 devices */
if ((chdev = ipm_Open("ipmB1C1", 0)) == -1) {
printf("Cannot open channel ipmB1C1. errno = %d", errno);
exit(1);
}

/* Get Device Information */
if (ipm_GetCTInfo(chdev, &ct_devinfo, EV_SYNC) == -1) {
printf("Error message = %s", ATDV_ERRMSGP(chdev));
exit(1);
}

printf("%s Product Id = 0x%x, Family = %d, Mode = %d, Network = %d, Bus
       mode = %d, Encoding = %d", ATDV_NAMEP(chdev), ct_devinfo.ct_prodid,
       ct_devinfo.ct_devfamily, ct_devinfo.ct_devmode, ct_devinfo.ct_nettype,
       ct_devinfo.ct_busmode, ct_devinfo.ct_busencoding);
}
```

See Also

- **ipm_Open( )**

ipm_GetLocalMediaInfo( )

Name: int ipm_GetLocalMediaInfo(nDeviceHandle, *pMediaInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_MEDIA_INFO *pMediaInfo	• pointer to media information structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetLocalMediaInfo( )** function retrieves properties for the local media channel. This function retrieves the local IP address and port information for the audio and/or video RTP/RTCP ports or T.38 port associated with the specified IP channel. These properties are assigned during firmware download.

The combination of the eMediaType field (in the IPM_MEDIA data structure pointed to by MediaData[ ], which in turn is a field of the IPM_MEDIA_INFO data structure pointed to by **pMediaInfo**) and the unCount field (in the IPM_MEDIA_INFO data structure pointed to by **pMediaInfo**) specifies the media type(s) for which port information is to be retrieved. The allowed combinations are as follows:

- To retrieve RTP port information for both audio and video in a multimedia session, set the eMediaType fields to MEDIATYPE_AUDIO_LOCAL_RTP_INFO and MEDIATYPE_VIDEO_LOCAL_RTP_INFO respectively and unCount to 2. See the code example.
- To retrieve RTP port information for a video only session, set the eMediaType field to MEDIATYPE_VIDEO_LOCAL_RTP_INFO and unCount to 1.
- To retrieve RTP port information for an audio only session, set the eMediaType field to MEDIATYPE_AUDIO_LOCAL_RTP_INFO and unCount to 1.
- To retrieve T.38 fax port information, set the eMediaType field to MEDIATYPE_LOCAL_UDPTL_T38_INFO and unCount to 1.

For audio and/or video media types, only RTP information needs to be queried. RTCP information is automatically provided when querying for RTP.

Note: The RTCP port number is the RTP port number + 1.

Note: It is not possible to retrieve T.38 fax port information together with audio and/or video port information.

To run this function asynchronously, set **mode** to EV_ASYNC. The function returns 0 if successful and the application must wait for the IPMEV_GET_LOCAL_MEDIA_INFO event. Once the event has been returned, use Dialogic® Standard Runtime Library (SRL) functions to retrieve IPM_MEDIA_INFO structure fields.

To run this function synchronously, set **mode** to EV_SYNC. The function returns 0 if successful and the IPM_MEDIA_INFO structure fields will be filled in.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pMediaInfo	pointer to structure that contains local IP address and IP channel port information for audio and/or video RTP / RTCP ports or T.38 port See the IPM_MEDIA_INFO data structure page for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_LOCAL_MEDIA_INFO
: Indicates successful completion, that is, local media information was received. Once the event has been returned, use Dialogic® SRL functions to retrieve IPM_MEDIA_INFO structure fields.

IPMEV_ERROR
: Indicates that the function failed.

Cautions

- For the Dialogic® Multimedia Platform for AdvancedTCA, this function returns an invalid state error if a coder is not reserved prior to calling **ipm_GetLocalMediaInfo( )**. All audio and/or video coders must be reserved using the **dev_ReserveResourceEx( )** Dialogic® Device Management Library function. However, this restriction does not apply when using a fax (T.38) media type.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
: Invalid parameter

EIPM_INTERNAL
: Internal error

EIPM_INV_MODE
: Invalid mode

EIPM_INV_STATE
: Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Get the local IP information for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    IPM_MEDIA_INFO MediaInfo;
    MediaInfo.unCount = 2;
    MediaInfo.MediaData[0].eMediaType = MEDIATYPE_VIDEO_LOCAL_RTP_INFO;
    // MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_UDPTL_T38_INFO;
    MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_LOCAL_RTP_INFO;


    if(ipm_GetLocalMediaInfo(nDeviceHandle, &MediaInfo, EV_ASYNC) == -1)
    {
        printf("ipm_GetLocalMediaInfo failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
    /*
    .
    .
    . Continue processing
    .
    .
    */
}

void CheckEvent()
{
    unsigned int i;
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    void* pVoid = sr_getevtdatap();
    IPM_MEDIA_INFO* pMediaInfo;
```

```
    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_GetLocalMediaInfo */
        case IPMEV_GET_LOCAL_MEDIA_INFO:
            printf("Received IPMEV_GET_LOCAL_MEDIA_INFO for device name = %s\n",
                ATDV_NAMEP(nDeviceID));
            pMediaInfo = (IPM_MEDIA_INFO*)pVoid;
            for(i=0; i<pMediaInfo->unCount; i++)
            {
               switch(pMediaInfo->MediaData[i].eMediaType)
               {
                  case MEDIATYPE_VIDEO_LOCAL_RTP_INFO:
                     printf("MediaType=MEDIATYPE_VIDEO_LOCAL_RTP_INFO\n");
                     printf("PortId=%d\n”,pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
                     printf("IP=%s\n”,pMediaInfo>MediaData[i].mediaInfo.PortInfo.cIPAddress);
                     break;
                  case MEDIATYPE_VIDEO_LOCAL_RTCP_INFO:
                     printf("MediaType=MEDIATYPE_VIDEO_LOCAL_RTCP_INFO\n");
                     printf("PortId=%d\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
                     printf("IP=%s\n",pMediaInfo>MediaData[i].mediaInfo.PortInfo.cIPAddress);
                     break;
                  case MEDIATYPE_AUDIO_LOCAL_RTP_INFO:
                     printf("MediaType=MEDIATYPE_AUDIO_LOCAL_RTP_INFO\n");
                     printf("PortId=%d\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
                     printf("IP=%s\n",pMediaInfo>MediaData[i].mediaInfo.PortInfo.cIPAddress);
                     break;
                  case MEDIATYPE_AUDIO_LOCAL_RTP_INFO:
                     printf("MediaType=MEDIATYPE_AUDIO_LOCAL_RTP_INFO\n");
                     printf("PortId=%d\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
                     printf("IP=%s\n",pMediaInfo>MediaData[i].mediaInfo.PortInfo.cIPAddress);
                     break;
                }
            break;

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

See Also

None

ipm_GetParm( )

Name: int ipm_GetParm(nDeviceHandle, *pParmInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_PARM_INFO *pParmInfo	• pointer to parameter info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetParm( )** function retrieves the current value of a parameter.

To run this function asynchronously, set mode to EV_ASYNC. The function returns 0 if successful and the application must wait for the IPMEV_GETPARM event. Once the event has been returned, use Dialogic® Standard Runtime Library (SRL) functions to retrieve parameter values.

To run this function synchronously, set mode to EV_SYNC. The function returns 0 if successful and the IPM_PARM_INFO structure fields will be filled in with the retrieved parameter information.

Parameter	Description
nDeviceHandle	handle of the IP media device
***pParmInfo**	pointer to structure that contains IP channel parameter values See the IPM_PARM_INFO data structure page for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_PARM
Indicates successful completion. Use Dialogic® SRL functions to retrieve IPM_PARM_INFO structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    ASSUMPTION: A valid nDeviceHandle was obtained from prior
    call to ipm_Open().
    */
    IPM_PARM_INFO ParmInfo;
    unsigned long ulParmValue = 0;
    ParmInfo.eParm = PARMCH_ECHOTAIL;
    ParmInfo.pvParmValue = &ulParmValue;
    if (ipm_GetParm(nDeviceHandle, &ParmInfo, EV_ASYNC)==-1)
    {
        printf("ipm_GetParm failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    ulParmValue = 0;
    ParmInfo.eParm = PARMCH_ECHOTAIL;
```

```
    if (ipm_GetParm(nDeviceHandle, &ParmInfo, EV_SYNC)==-1)
    {
        printf("%s: ipm_GetParm failed..exiting..!!!\n", ATDV_NAMEP(nDeviceHandle));
    }
    else
    {
        printf("%s: ipm_GetParm(parm=0x%x,value=0x%x) ok %\n", ATDV_NAMEP(nDeviceHandle),
            ParmInfo.eParm, ulParmValue );
    }

    /*
    .
    .
    . continue
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_PARM_INFO* pParmInfo;

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_GetQoSAlarmStatus */
        case IPMEV_GET_PARM:
            pParmInfo = (IPM_PARM_INFO*) pVoid;
            printf("Received IPMEV_GETPARM for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            printf("%s: parm=0x%x, ok %\n", ATDV_NAMEP(nDeviceID),
                pParmInfo->eParm);
            break;

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

See Also

- **ipm_SetParm( )**

ipm_GetQoSAlarmStatus( )

Name: int ipm_GetQoSAlarmStatus(nDeviceHandle, *pQoSAlarmInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_QOS_ALARM_STATUS *pQoSAlarmInfo	• pointer to QoS alarm status structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetQoSAlarmStatus( )** function retrieves the ON/OFF state of Quality of Service (QoS) alarms that report the status of a media channel.

This function returns the status of media channel QoS alarms that are enumerated in eIPM_QOS_TYPE. This function does not return the status of board-level alarms.

Use **ipm_ResetQoSAlarmStatus( )** to reset the QoS alarm state.

Parameter	Description
nDeviceHandle	handle of the IP Media channel device
pQoSAlarmInfo	pointer to structure that contains alarm identifier and alarm status values See IPM_QOS_ALARM_STATUS for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_QOS_ALARM_STATUS
Indicates successful completion. Use SRL functions to retrieve IPM_QOS_ALARM_STATUS structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>
void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
   int nDeviceHandle;
   // Register event handler function with srl
   sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

   /*
   .
   .
    Main Processing
   .
   .
   */
   /*
    Query the alarm status for IP device handle, nDeviceHandle.
    ASSUMPTION:  A valid nDeviceHandle was obtained from prior call to ipm_Open().
   */
   IPM_QOS_ALARM_STATUS AlarmStatus;
   AlarmStatus.unAlarmCount = 1;
   AlarmStatus.QoSData[0].eQoSType = QOSTYPE_LOSTPACKETS;

   if(ipm_GetQoSAlarmStatus(nDeviceHandle, &AlarmStatus, EV_ASYNC) == -1)
   {
      printf("ipm_GetQoSAlarmStatus failed for device name %s with error = %d\n",
             ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
      /*
      .
      .
      Perform Error Processing
      .
      .
      */
   }
```

```
    /*
    .
    .
    continue
    .
    .
    */
}

void CheckEvent()
{
    int i;
    int nEventType = sr_getevttype();
    int nDeviceID  = sr_getevtdev();
    void* pVoid    = sr_getevtdatap();
    IPM_QOS_ALARM_STATUS* pmyAlarmStatus;
    switch(nEventType)
    {
        /*
        .
        .
        Other events
        .
        .
        */
        /* Expected reply to ipm_GetQoSAlarmStatus */
        case IPMEV_GET_QOS_ALARM_STATUS:
            pmyAlarmStatus = (IPM_QOS_ALARM_STATUS*)pVoid;
            printf("Received IPMEV_GET_QOS_ALARM_STATUS for device = %s\n",
                   ATDV_NAMEP(nDeviceID));
            for(i=0; i<pmyAlarmStatus->unAlarmCount; ++i)
            {
                switch(pmyAlarmStatus->QoSData[i].eQoSType)
                {
                    case QOSTYPE_LOSTPACKETS:
                        printf("  LOSTPACKETS = %d\n",l_myAlarmStatus.QoSData[i].eAlarmState);
                        break;
                    case QOSTYPE_JITTER:
                        printf("  JITTER = %d\n",l_myAlarmStatus.QoSData[i].eAlarmState);
                        break;
                }
            }
            break;
        default:
            printf("Received unknown event = %d  for device = %s\n",
                   nEventType, ATDV_NAMEP(nDeviceID));
            break;
    }
}
```

See Also

- **ipm_ResetQoSAlarmStatus( )**

ipm_GetQoSThreshold( )

Name: int ipm_GetQoSThreshold(nDeviceHandle, *pQoSThresholdInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_QOS_THRESHOLD_INFO *pQoSThresholdInfo	• pointer to QoS alarm threshold structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetQoSThreshold( )** function retrieves alarm threshold settings for Quality of Service (QoS) alarms that report the status of media channels.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pQoSThresholdInfo	pointer to IPM_QOS_THRESHOLD_INFO structure which contains one or more IPM_QOS_THRESHOLD_DATA structures
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_QOS_THRESHOLD_INFO
Indicates successful completion. Use SRL functions to retrieve IPM_QOS_THRESHOLD_INFO structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- The IPM_QOS_THRESHOLD_INFO structure specifies the QoS Alarm Identifier thresholds. The application may use this structure to get statistics for only specified QoS types. Use SRL functions to retrieve IPM_QOS_THRESHOLD_INFO structure fields.
- If **ipm_GetQoSThreshold( )** is called synchronously, the IPM_QOS_THRESHOLD_INFO structure is both an input and output parameter. If **ipm_GetQoSThreshold( )** is called

asynchronously, the structure is used only as an input parameter. To retrieve all the QoS threshold settings, in both synchronous and asynchronous modes, set the **unCount** field in IPM_QOS_THRESHOLD_INFO structure to 0.

- This function does not apply to board-level alarms because these alarms do not have settable threshold values.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
   int nDeviceHandle;
   // Register event handler function with srl
   sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

   /*
   .
   .
   Main Processing
   .
   .
   */

   /*
   Query the alarm threshold settings for IP device handle, nDeviceHandle.
   ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
   */
   IPM_QOS_THRESHOLD_INFO myThresholdInfo;
   myThresholdInfo.unCount = 0;
   if(ipm_GetQoSThreshold(nDeviceHandle, &myThresholdInfo, EV_ASYNC) == -1)
   {
      printf("ipm_GetQoSThreshold failed for device name = %s with error = %d\n",
          ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
```

```
      /*
      .
      .
      Perform Error Processing
      .
      .
      */
   }

   /*
   .
   .
   . continue
   .
   .
   */
}

void CheckEvent()
{
   unsigned int i;
   int nEventType = sr_getevttype();
   int nDeviceID = sr_getevtdev();
   void* pVoid = sr_getevtdatap();
   IPM_QOS_THRESHOLD_INFO* pThresholdInfo;

   switch(nEventType)
   {
      /*
      .
      .
      . Other events
      .
      .
      */

      /* Expected reply to ipm_GetQoSThreshold */
      case IPMEV_GET_QOS_THRESHOLD_INFO:
         pThresholdInfo = (IPM_QOS_THRESHOLD_INFO*)pVoid;
         printf("Received IPMEV_GET_QOS_THRESHOLD_INFO for device = %s\n",
             ATDV_NAMEP(nDeviceID));
         for(i=0; i<pThresholdInfo->unCount; ++i)
         {
            switch(pThresholdInfo->QoSThresholdData[i].eQoSType)
            {

               case QOSTYPE_LOSTPACKETS:
                  printf("QOSTYPE_LOSTPACKETS\n");
                  printf("unTimeInterval = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unTimeInterval);
                  printf("unDebounceOn = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unDebounceOn);
                  printf("unDebounceOff = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unDebounceOff);
                  printf("unFaultThreshold = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unFaultThreshold);
                  printf("unPercentSuccessThreshold = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unPercentSuccessThreshold);
                  printf("unPercentFailThreshold = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unPercentFailThreshold);
                  break;

               case QOSTYPE_JITTER:
                  printf("QOSTYPE_JITTER\n");
                  printf("unTimeInterval = %d\n",
                         pThresholdInfo->QoSThresholdData[i].unTimeInterval);
                  printf("unDebounceOn = %d\n",
```

```
                        pThresholdInfo->QoSThresholdData[i].unDebounceOn);
                    printf("unDebounceOff = %d\n",
                        pThresholdInfo->QoSThresholdData[i].unDebounceOff);
                    printf("unFaultThreshold = %d\n",
                        pThresholdInfo->QoSThresholdData[i].unFaultThreshold);
                    printf("unPercentSuccessThreshold = %d\n",
                        pThresholdInfo->QoSThresholdData[i].unPercentSuccessThreshold);
                    printf("unPercentFailThreshold = %d\n",
                        pThresholdInfo->QoSThresholdData[i].unPercentFailThreshold);
                    break;
                }
            }
            break;

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

See Also

- **ipm_SetQoSThreshold( )**

ipm_GetSessionInfo( )

Name: int ipm_GetSessionInfo(nDeviceHandle, *pSessionInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_SESSION_INFO *pSessionInfo	• pointer to session info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetSessionInfo( )** function retrieves QoS and RTCP statistics for media session, if one is in progress; otherwise, it retrieves statistics for the previous session.

If a media session has been initiated by calling **ipm_StartMedia( )**, the data returned by **ipm_GetSessionInfo( )** is for the current session. If **ipm_GetSessionInfo( )** is called between media sessions—that is, after **ipm_Stop( )** terminates the session and before **ipm_StartMedia( )** is called to start a new session—the data returned is for that previous media session.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pSessionInfo	pointer to structure that contains Quality of Service (QoS) information about the previous IP session. This parameter can be NULL if the function is called in the asynchronous mode. See IPM_SESSION_INFO for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_SESSION_INFO
Indicates successful completion; that is, the structure containing session statistics was filled in. Use SRL functions to retrieve IPM_SESSION_INFO structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- The application can call **ipm_GetQoSAlarmStatus( )** to retrieve alarm information for the current session.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Get the current session information for IP device handle, nDeviceHandle.
    ASSUMPTION: nDeviceHandle was obtained from a prior call to ipm_Open().
    Also, ipm_StartMedia() was successfully called some time earlier.
    */
    if(ipm_GetSessionInfo(nDeviceHandle, NULL, EV_ASYNC) == -1)
    {
        printf("ipm_GetSessionInfo failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
```

```
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    . Continue processing
    .
    .
    */
}

void CheckEvent()
{
    unsigned int i;
    IPM_SESSION_INFO* pIPSessionInfo;
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    void* pVoid = sr_getevtdatap();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_GetSessionInfo */
        case IPMEV_GET_SESSION_INFO:
            pIPSessionInfo = (IPM_SESSION_INFO*)pVoid;
            printf("Received IPMEV_GET_SESSION_INFO for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            printf("RtcpInfo.unLocalSR_TimeStamp=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalSR_TimeStamp);
            printf("RtcpInfo.unLocalSR_TxPackets=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalSR_TxPackets);
            printf("RtcpInfo.unLocalSR_TxOctets=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalSR_TxOctets);
            printf("RtcpInfo.unLocalSR_SendIndication=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalSR_SendIndication);
            printf("RtcpInfo.unLocalRR_FractionLost=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalRR_FractionLost);
            printf("RtcpInfo.unLocalRR_CumulativeLost=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalRR_CumulativeLost);
            printf("RtcpInfo.unLocalRR_SeqNumber=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalRR_SeqNumber);
            printf("RtcpInfo.unLocalRR_ValidInfo=%d\n",
                pIPSessionInfo->RtcpInfo.unLocalRR_ValidInfo);
            printf("RtcpInfo.unRemoteSR_TimeStamp=%d\n",
                pIPSessionInfo->RtcpInfo.unRemoteSR_TimeStamp);
            printf("RtcpInfo.unRemoteSR_TxPackets=%d\n",
                pIPSessionInfo->RtcpInfo.unRemoteSR_TxPackets);
            printf("RtcpInfo.unRemoteSR_TxOctets=%d\n",
                pIPSessionInfo->RtcpInfo.unRemoteSR_TxOctets);
            printf("RtcpInfo.unRemoteSR_SendIndication=%d\n",
                pIPSessionInfo->RtcpInfo.unRemoteSR_SendIndication);
            printf("RtcpInfo.unRemoteRR_FractionLost=%d\n",
                pIPSessionInfo->RtcpInfo.unRemoteRR_FractionLost);
```

```
                printf("RtcpInfo.unRemoteRR_CumulativeLost=%d\n",
                    pIPSessionInfo->RtcpInfo.unRemoteRR_CumulativeLost);
                printf("RtcpInfo.unRemoteRR_SeqNumber=%d\n",
                    pIPSessionInfo->RtcpInfo.unRemoteRR_SeqNumber);
                printf("RtcpInfo.unRemoteRR_ValidInfo=%d\n",
                    pIPSessionInfo->RtcpInfo.unRemoteRR_ValidInfo);

                for(i = 0; i< pIPSessionInfo->unQoSInfoCount; ++i)
                {
                    printf("Session QOS Type=%d\n", pIPSessionInfo->QoSInfo[i].eQoSType);
                    printf("Session QOS Data=%d\n", pIPSessionInfo->QoSInfo[i].unData);
                }
                break;

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

See Also

- **ipm_GetQoSAlarmStatus( )**
- **ipm_StartMedia( )**

ipm_GetXmitSlot( )

Name: int ipm_GetXmitSlot(nDeviceHandle, *pTimeslotInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
SC_TSINFO *pTimeslotInfo	• pointer to time slot info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_GetXmitSlot( )** function returns TDM time slot information for an IP channel.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pTimeslotInfo	pointer to structure that describes the time slot number, time slot type, and bus encoding format. This parameter can be NULL if the function is called in the asynchronous mode. See SC_TSINFO for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_GET_XMITTS_INFO
Indicates successful completion. Use SRL functions to retrieve SC_TSINFO structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Get the timeslot information for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_GetXmitSlot(nDeviceHandle, NULL, EV_ASYNC) == -1)
    {
        printf("ipm_GetXmitSlot failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
```

```
    /*
    .
    .
    . continue
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    SC_TSINFO* pTimeSlotInfo;

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_GetXmitSlot */
        case IPMEV_GET_XMITTS_INFO:
            pTimeSlotInfo = (SC_TSINFO*)pVoid;
            printf("Received IPMEV_GET_XMITTS_INFO for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            printf("Timeslot number %d\n", *(pTimeSlotInfo->sc_tsarrayp));
            break;

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

See Also

None

ipm_InitResponseSend( )

Name: int ipm_InitResponseSend (a_nDeviceHandle, pInitRsp)

Inputs:

int a_nDeviceHandle	• SRL handle of the IPM device
const PIPM_NBUP_INIT_RESPONSE a_pInitRsp	• structure describing the Nb UP response

Returns: 0 on success.
-1 on error

Includes: ipmlib.h

Category: Media Session

Mode: asynchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_InitResponseSend( )** function attempts to send a response to an IP session (for example, Nb UP session) invitation. The **ipm_StartMedia( )** function must have been previously called with the appropriate connection type, such as Nb UP.

Parameter	Description
a_nDeviceHandle	SRL handle of the IP media device
a_pInitRsp	structure describing the Nb UP response. See IPM_NBUP_INIT_RESPONSE for details.

Events

If the function returns 0, it can generate any of the following events:

IPMEV_INIT_RESPONSE_SEND
: Indicates successful completion of sending the initialization response message. It does not indicate whether the event was received by the remote party. If IPMEV_INIT_RESPONSE_SEND is received, one of the following unsolicited events is subsequently reported:
  - IPMEV_INIT_COMPLETE – Unsolicited event reported when the Nb UP session has been successfully negotiated with the remote party. Both parties may now begin exchanging data.
  - IPMEV_INIT_FAILED – Unsolicited event reported upon encountering an unexpected error during initialization. Event Data: IPM_NBUP_INIT_FAILURE structure.

IPMEV_ERROR
: Indicates that the function failed.

Cautions

None.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_BUSY
Channel is busy

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "srllib.h"
#include "dxxxlib.h"

#include "ipmlib.h"

static int g_hIPM = -1;
static IPM_MEDIA_INFO g_media, g_localMedia;

/*****************************************************************************
* Dialogic API function callers
*****************************************************************************/
bool GetLocalMediaInfo()
{
   g_localMedia.unCount = 1;
   g_localMedia.MediaData[0].eMediaType =
         MEDIATYPE_NBUP_LOCAL_RTP_INFO;

   if( ipm_GetLocalMediaInfo(
         g_hIPM,
         &g_localMedia,
         EV_ASYNC ) == -1 )
   {
      printf("ipm_GetLocalMediaInfo failed\n");
      return false;
   }

   return true;
}
```

```
bool StartMedia()
{
   g_media.unCount = 2;

   // Set up remote RTP Media Info
   g_media.MediaData[0].eMediaType = MEDIATYPE_NBUP_REMOTE_RTP_INFO;
   g_media.MediaData[0].mediaInfo.PortInfo.unPortId =
      g_localMedia.MediaData[0].mediaInfo.PortInfo.unPortId;
   strcpy(
      g_media.MediaData[0].mediaInfo.PortInfo.cIPAddress,
      g_localMedia.MediaData[0].mediaInfo.PortInfo.cIPAddress);

   // Set up NBUP info
   g_media.MediaData[1].eMediaType = MEDIATYPE_NBUP_PROFILE_INFO;
   INIT_IPM_NBUP_PROFILE_INFO(
         &(g_media.MediaData[1].mediaInfo.NBUPProfileInfo));
   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.eProfileType =
         NBUP_PROFILE_3G324M;

   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.ucTxPLT = 111;
   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.ucRxPLT = 111;

   if (ipm_StartMedia(
         g_hIPM,
         &g_media,
         DATA_IP_TDM_BIDIRECTIONAL,
         EV_ASYNC ) == -1 )
   {
      printf("ipm_StartMedia failed\n");
      return false;
   }

   return true;
}

bool InitResponseSend()
{
   IPM_INIT_RESPONSE initResp;
   IPM_NBUP_INIT_RESPONSE nbupResp;

   // Initialize the main structure
   INIT_IPM_INIT_RESPONSE(&initResp);
   initResp.eProtocol = RTP_PROTOCOL_NBUP;
   initResp.data.pNBUP = &nbupResp;

   // Initialize the NBUP specific structure with an ACK response
   INIT_IPM_NBUP_INIT_RESPONSE(&nbupResp);
   nbupResp.eResponse = NBUP_INIT_RESPONSE_ACK;

   if (ipm_InitResponseSend(g_hIPM, &initResp) == -1)
   {
      printf("ipm_InitResponseSend failed\n");
      return false;
   }

   return true;
}

bool EnableEvents()
{
   const int nEvents = 4;
   eIPM_EVENT events[nEvents] =
   {
      EVT_INIT_RECEIVED,
      EVT_PROCEDURE_DONE,
      EVT_SEND_FAILED,
```

```
      EVT_NACK_SENT
   };

   if (ipm_EnableEvents(
         g_hIPM,
         events,
         nEvents,
         EV_ASYNC) == -1)
   {
        printf("ipm_EnableEvents failed\n");
        return false;
   }

   return true;
}

/*****************************************************************************
Event handler for Inbound IPM device
NOTE: Returning 1 will cause waitevt(-1) to stop blocking
      and exit the state machine.
*****************************************************************************/
long IpmEventHandlerInbound( unsigned long evthandle )
{
   // Get the event info from SRL
   int evttype = sr_getevttype();
   int evtdev = sr_getevtdev();

   if (evtdev == -1) return 0;

   switch( evttype )
   {
   case IPMEV_EVENT_ENABLED:
      printf("IPMEV_EVENT_ENABLED event received.\n");

      // Get local Media Info to be used in ipm_StartMedia
      if (!GetLocalMediaInfo())
      {
         return 1;
      }
      break;

   case IPMEV_GET_LOCAL_MEDIA_INFO:
      printf("IPMEV_GET_LOCAL_MEDIA_INFO event received.\n");
      {
         // Search for the local media info in the returned data structure
         IPM_MEDIA_INFO *pMediaInfo = (IPM_MEDIA_INFO *)sr_getevtdatap();
         for (unsigned int ii = 0; ii < pMediaInfo->unCount; ii++)
         {
            switch(pMediaInfo->MediaData[ii].eMediaType)
            {
            case MEDIATYPE_NBUP_LOCAL_RTP_INFO:
               // Found local media info in the returned event, store it away.
               memcpy(&(g_localMedia.MediaData[0]), &(pMediaInfo->MediaData[ii]), sizeof
(IPM_MEDIA));

               // Call StartMedia
               if (!StartMedia())
               {
                  return 1;
               }

            default:
               printf("Unexpected media type %d found.\n", pMediaInfo-
>MediaData[ii].eMediaType);
               break;
            }
         }
```

```
      }
      break;

   case IPMEV_STARTMEDIA:
      printf("IPMEV_STARTMEDIA event received.\n");

      // Sit and wait for an inbound NBUP session request
      break;

   case IPMEV_INIT_RECEIVED:
      printf("IPMEV_INIT_RECEIVED event received.");
      {
         // Check to see what type of invite we received
         PIPM_INIT_RECEIVED pRecv = (PIPM_INIT_RECEIVED)sr_getevtdatap();

         if (!pRecv)
         {
            printf("Invalid pointer returned from sr_getevtdatap()");
            return 1;
         }

         if (pRecv->eProtocol == RTP_PROTOCOL_NBUP)
         {
            // We've received an NBUP invitation.  Send an ACK to start the session
            if (!InitResponseSend())
            {
               return 1;
            }
         }
         else
         {
            printf("Unsupported protocol (%d) in init request.", pRecv->eProtocol);
            return 1;
         }
      }
      break;

   case IPMEV_INIT_RESPONSE_SEND:
      printf("IPMEV_INIT_RESPONSE_SEND event received.");
      // Just the termination event for ipm_InitResponseSend
      break;

   case IPMEV_INIT_COMPLETE:
      printf("IPMEV_INIT_COMPLETE event received"
            " - NBUP CONNECTION SUCCESSFUL\n");

      // This means all has passed and we have an NBUP session.
      // Just exit out of our sample scenario here
      return 1;

   case IPMEV_INIT_FAILED:
      // Something prevented us from connecting
      printf("IPMEV_INIT_FAILED event received.\n");
      {
#ifdef WIN32
         PIPM_INIT_FAILURE pInfo = (PIPM_INIT_FAILURE)sr_getevtdatap(g_hIPM);
#else
         PIPM_INIT_FAILURE pInfo = (PIPM_INIT_FAILURE)sr_getevtdatap();
#endif
         if (pInfo)
         {
            if (pInfo->eProtocol == RTP_PROTOCOL_NBUP)
            {
               PIPM_NBUP_INIT_FAILURE pNBUP = (PIPM_NBUP_INIT_FAILURE)(pInfo->data.pNBUP);
               printf(
                  "Error cause: 0x%X\n",
                  pNBUP->unCause);
```

```
            }
            else
            {
               printf("Unexpected protocol %d found.\n", pInfo->eProtocol);
            }
         }
         else
         {
            printf("Bad pointer returned from sr_getevtdatap()\n");
         }
      }
      return 1;

   case IPMEV_ERROR:
      printf("IPMEV_ERROR %d: %s\n", ATDV_LASTERR(g_hIPM), ATDV_ERRMSGP(g_hIPM));
      return 1;

   default:
      printf("WARNING: Unknown event %d received.\n", evttype );
      break;
   } // end switch

   return 0;
}

/*************************************************************************
MAIN procedure
*************************************************************************/
void main()
{
   g_hIPM = ipm_Open("ipmB1C1\n", NULL, EV_SYNC );
   if( g_hIPM == -1 )
   {
      printf( "ipm_Open() failed.\n" );
      exit( 1 );
   }

   if (sr_enbhdlr( g_hIPM, EV_ANYEVT, IpmEventHandlerInbound ) == -1 )
   {
      printf( "sr_enbhdlr() failed.\n" );
      exit( 1 );
   }

   // Enable the NBUP events and kick off the state machine
   if (!EnableEvents())
   {
      exit( 1 );
   }

   // Wait for complete
   sr_waitevt(-1);

   // Shutdown
   if (ipm_Stop(g_hIPM, STOP_ALL, EV_SYNC == -1))
   {
      printf("ipm_Stop failed.\n");
   }

   if( sr_dishdlr( g_hIPM, EV_ANYEVT, IpmEventHandlerInbound ) == -1 )
   {
      printf( "sr_dishdlr() failed.\n" );
   }
```

```
    if( ipm_Close( g_hIPM, NULL ) == -1 )
    {
       printf( "ipm_Close() failed.\n" );
    }
}
```

See Also

- **ipm_InitSend( )**

ipm_InitSend( )

Name: int ipm_InitSend (a_nDeviceHandle, *a_pInitInfo)

Inputs: int a_nDeviceHandle
- SRL handle of the IPM device

const IPM_INIT_SEND a_pInitInfo
- structure describing the Nb UP initialization parameters

Returns: 0 on success
-1 on error

Includes: ipmlib.h

Category: Media Session

Mode: asynchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_InitSend( )** function attempts to send an IP protocol (for example, Nb UP) message to a remote party. The **ipm_StartMedia( )** function must have been previously called with the appropriate connection type, such as Nb UP.

Parameter	Description
a_nDeviceHandle	SRL handle of the IP media device
a_pInitInfo	structure describing the Nb UP initialization parameters. See IPM_INIT_SEND for details.

Events

If the function returns 0, it can generate any of the following events:

IPMEV_INIT_SEND
Indicates successful completion of sending the initialization message. It does not indicate whether the event was received by remote party or what the response was. If IPMEV_INIT_SEND is received, one of the following unsolicited events is subsequently reported:
- IPMEV_INIT_COMPLETE – Unsolicited event reported when the Nb UP session has been successfully negotiated with the remote party. Both parties may now begin exchanging data.
- IPMEV_INIT_FAILED – Unsolicited event reported upon encountering an unexpected error or a negative response (NACK) from the remote party. Event Data: IPM_INIT_FAILURE structure.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** functions to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_BUSY
Channel is busy

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "srllib.h"
#include "dxxxlib.h"

#include "ipmlib.h"

static int g_hIPM = -1;
static IPM_MEDIA_INFO g_media, g_localMedia;

/*****************************************************************************
* Dialogic API function callers
*****************************************************************************/
bool GetLocalMediaInfo()
{
   g_localMedia.unCount = 1;
   g_localMedia.MediaData[0].eMediaType =
         MEDIATYPE_NBUP_LOCAL_RTP_INFO;

   if( ipm_GetLocalMediaInfo(
         g_hIPM,
         &g_localMedia,
         EV_ASYNC ) == -1 )
   {
      printf("ipm_GetLocalMediaInfo failed\n");
      return false;
   }

   return true;
}
```

```
bool StartMedia()
{
   g_media.unCount = 2;

   // Set up remote RTP Media Info
   g_media.MediaData[0].eMediaType = MEDIATYPE_NBUP_REMOTE_RTP_INFO;
   g_media.MediaData[0].mediaInfo.PortInfo.unPortId =
      g_localMedia.MediaData[0].mediaInfo.PortInfo.unPortId;
   strcpy(
      g_media.MediaData[0].mediaInfo.PortInfo.cIPAddress,
      g_localMedia.MediaData[0].mediaInfo.PortInfo.cIPAddress);

   // Set up NBUP info
   g_media.MediaData[1].eMediaType = MEDIATYPE_NBUP_PROFILE_INFO;
   INIT_IPM_NBUP_PROFILE_INFO(
         &(g_media.MediaData[1].mediaInfo.NBUPProfileInfo));
   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.eProfileType =
         NBUP_PROFILE_3G324M;

   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.ucTxPLT = 111;
   g_media.MediaData[1].mediaInfo.NBUPProfileInfo.ucRxPLT = 111;

   if (ipm_StartMedia(
         g_hIPM,
         &g_media,
         DATA_IP_TDM_BIDIRECTIONAL,
         EV_ASYNC ) == -1 )
   {
      printf("ipm_StartMedia failed\n");
      return false;
   }

   return true;
}

bool InitSend()
{
   // Initiate the NBUP session
   IPM_INIT_SEND initSend;
   IPM_NBUP_INIT_SEND nbupSend;

   // Initialize the main structure
   INIT_IPM_INIT_SEND(&initSend);
   initSend.eProtocol = RTP_PROTOCOL_NBUP;
   initSend.data.pNBUP = &nbupSend;

   // Initialize the NBUP specific structure
   INIT_IPM_NBUP_INIT_SEND(&nbupSend, 1, 1);

   // Init function handles memory allocation for us
   nbupSend.pRFCIs[0].ucID = (unsigned char)0x44;
   nbupSend.pRFCIs[0].pSubFlows[0].eFlowSize =
         NBUP_FLOW_SIZE_320_BITS;

   if (ipm_InitSend(g_hIPM, &initSend) == -1)
   {
      // De-allocate memory allocated for us
      FREE_IPM_NBUP_INIT_SEND(&nbupSend);

      printf("ipm_InitSend failed\n");
      return false;
   }

   // De-allocate memory allocated for us
   FREE_IPM_NBUP_INIT_SEND(&nbupSend);
```

```
   return true;
}

bool EnableEvents()
{
   const int nEvents = 4;
   eIPM_EVENT events[nEvents] =
   {
      EVT_INIT_RECEIVED,
      EVT_PROCEDURE_DONE,
      EVT_SEND_FAILED,
      EVT_NACK_SENT
   };

   if (ipm_EnableEvents(
         g_hIPM,
         events,
         nEvents,
         EV_ASYNC) == -1)
   {
        printf("ipm_EnableEvents failed\n");
        return false;
   }

   return true;
}

/*************************************************************************
Event handler for Outbound IPM device
NOTE: Returning 1 will cause waitevt(-1) to stop blocking
      and exit the state machine.
*************************************************************************/
long IpmEventHandlerOutbound( unsigned long evthandle )
{
   // Get the event info from SRL
   int evttype = sr_getevttype();
   int evtdev = sr_getevtdev();

   if (evtdev == -1) return 0;

   switch( evttype )
   {
   case IPMEV_EVENT_ENABLED:
      printf("IPMEV_EVENT_ENABLED event received.\n");

      // Get local Media Info to be used in ipm_StartMedia
      if (!GetLocalMediaInfo())
      {
         return 1;
      }
      break;

   case IPMEV_GET_LOCAL_MEDIA_INFO:
      printf("IPMEV_GET_LOCAL_MEDIA_INFO event received.\n");
      {
         // Search for the local media info in the returned data structure
         IPM_MEDIA_INFO *pMediaInfo = (IPM_MEDIA_INFO *)sr_getevtdatap();
         for (unsigned int ii = 0; ii < pMediaInfo->unCount; ii++)
         {
            switch(pMediaInfo->MediaData[ii].eMediaType)
            {
            case MEDIATYPE_NBUP_LOCAL_RTP_INFO:
               // Found local media info in the returned event, store it away.
               memcpy(&(g_localMedia.MediaData[0]), &(pMediaInfo->MediaData[ii]), sizeof
(IPM_MEDIA));

               // Call StartMedia
```

```
               if (!StartMedia())
               {
                  return 1;
               }

            default:
               printf("Unexpected media type %d found.\n", pMediaInfo-
>MediaData[ii].eMediaType);
               break;
            }
         }
      }
      break;

   case IPMEV_STARTMEDIA:
      printf("IPMEV_STARTMEDIA event received.\n");

      // Initiate the NBUP session
      if (!InitSend())
      {
         return 1;
      }
      break;

   case IPMEV_INIT_SEND:
      printf("IPMEV_INIT_SEND event received.\n");
      // Just the termination event for ipm_InitSend - take no action
      break;

   case IPMEV_INIT_COMPLETE:
      printf("IPMEV_INIT_COMPLETE event received"
            " - NBUP CONNECTION SUCCESSFUL\n");

      // This means all has passed and we have an NBUP session.
      // Just exit out of our sample scenario here
      return 1;

   case IPMEV_INIT_FAILED:
      // Something prevented us from connecting
      printf("IPMEV_INIT_FAILED event received.\n");
      {
#ifdef WIN32
         PIPM_INIT_FAILURE pInfo = (PIPM_INIT_FAILURE)sr_getevtdatap(g_hIPM);
#else
         PIPM_INIT_FAILURE pInfo = (PIPM_INIT_FAILURE)sr_getevtdatap();
#endif
         if (pInfo)
         {
            if (pInfo->eProtocol == RTP_PROTOCOL_NBUP)
            {
               PIPM_NBUP_INIT_FAILURE pNBUP = (PIPM_NBUP_INIT_FAILURE)(pInfo->data.pNBUP);
               printf(
                  "Error cause: 0x%X\n",
                  pNBUP->unCause);
            }
            else
            {
               printf("Unexpected protocol %d found.\n", pInfo->eProtocol);
            }
         }
         else
         {
            printf("Bad pointer returned from sr_getevtdatap()\n");
         }
      }
      return 1;
```

```
   case IPMEV_ERROR:
      printf("IPMEV_ERROR %d: %s\n", ATDV_LASTERR(g_hIPM), ATDV_ERRMSGP(g_hIPM));
      return 1;

   default:
      printf("WARNING: Unknown event %d received.\n", evttype );
      break;
   } // end switch

   return 0;
}

/*****************************************************************************
MAIN procedure
*****************************************************************************/
void main()
{
   g_hIPM = ipm_Open("ipmB1C1\n", NULL, EV_SYNC );
   if( g_hIPM == -1 )
   {
      printf( "ipm_Open() failed.\n" );
      exit( 1 );
   }

   if (sr_enbhdlr( g_hIPM, EV_ANYEVT, IpmEventHandlerOutbound ) == -1 )
   {
      printf( "sr_enbhdlr() failed.\n" );
      exit( 1 );
   }

   // Enable the NBUP events and kick off the state machine
   if (!EnableEvents())
   {
      exit( 1 );
   }

   // Wait for complete
   sr_waitevt(-1);

   // Shutdown
   if (ipm_Stop(g_hIPM, STOP_ALL, EV_SYNC == -1))
   {
      printf("ipm_Stop failed.\n");
   }

   if( sr_dishdlr( g_hIPM, EV_ANYEVT, IpmEventHandlerOutbound ) == -1 )
   {
      printf( "sr_dishdlr() failed.\n" );
   }

   if( ipm_Close( g_hIPM, NULL ) == -1 )
   {
      printf( "ipm_Close() failed.\n" );
   }
}
```

See Also

- **ipm_InitResponseSend( )**

ipm_Listen( )

Name: int ipm_Listen(nDeviceHandle, *pTimeslotInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
SC_TSINFO *pTimeslotInfo	• pointer to time slot info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_Listen( )** function connects an IP channel to a TDM time slot, enabling data to flow between the TDM time slot and the IP network or the host.

ipm_Listen( ) uses the information stored in the SC_TSINFO structure to connect the receive channel on the device to an available TDM bus time slot in the specified list of time slots. The time slot number is returned in the SC_TSINFO structure. The receive channel remains connected to the TDM bus time slot until **ipm_UnListen( )** is called or **ipm_Listen( )** is called with a different time slot.

If **ipm_Listen( )** is called to connect to a different TDM time slot, the firmware automatically breaks an existing connection and reconnects it to the new time slot. In this case, the application does not need to call the **ipm_UnListen( )** function.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pTimeslotInfo	pointer to structure that describes the time slot number, time slot type, and bus encoding format See SC_TSINFO for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_LISTEN
Indicates successful completion; that is, an IP channel was connected to the specified TDM time slot. This event does not return any data.

IPMEV_ERROR
Indicates the function failed.

Cautions

- The IP Media library allows **ipm_Listen( )** and **ipm_UnListen( )** to be called either synchronously or asynchronously. Other Dialogic libraries may not support asynchronous execution of the similar **xx_Listen** and **xx_UnListen** functions.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    SC_TSINFO IPTimeSlotInfo;
    long lTimeSlot;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Tell IP device handle, nDeviceHandle, to listen to timeslot 10.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    lTimeSlot = 10;
    IPTimeSlotInfo.sc_tsarrayp = &lTimeSlot;
    IPTimeSlotInfo.sc_numts = 1;
```

```
    if(ipm_Listen(nDeviceHandle, &IPTimeSlotInfo, EV_ASYNC) == -1)
    {
        printf("ipm_Listen failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    . Continue processing
    .
    .
    */
}

void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_Listen */
        case IPMEV_LISTEN:
            printf("Received IPMEV_LISTEN for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_UnListen( )**

ipm_ModifyMedia( )

Name: int ipm_ModifyMedia(nDeviceHandle, *pMediaInfo, eDirection, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_MEDIA_INFO *pMediaInfo	• pointer to media information structure
eIPM_DATA_DIRECTION eDirection	• data flow direction
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software

Description

The **ipm_ModifyMedia( )** function modifies various properties of an active media session. This function allows the application to modify the following media session properties:

- direction of the media stream
- remote IP address and port

For this function to complete successfully, the stream associated with the IP device must be in either active or suspended mode.

The media session properties are changed on the local endpoint as soon the function is called, and this may result in a perceptible artifact (for example, a click or a brief silence) until the remote endpoint makes the corresponding change. For example, if the coder is being changed by the function call, the local endpoint begins transmitting packets using the new coder and stops accepting packets that it receives which use the old coder as soon as the function executes.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pMediaInfo	pointer to structure that contains local channel RTP/RTCP ports and IP address information (or T.38 port and IP address information) See the IPM_MEDIA_INFO data structure page for details.

Parameter	Description
eDirection	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: • DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data • DATA_IP_SENDONLY – send data to the IP network but do not receive data • DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM • DATA_IP_INACTIVE – allow RTCP while blocking RTP or T.38 packets
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution

Termination Events

In asynchronous mode, the function returns 0 if the operation was initiated successfully. Completion of the operation is indicated by receipt of a termination event:

IPMEV_MODIFY_MEDIA
: Indicates successful completion; that is, modified media information was set and the session has been started.

IPMEV_MODIFY_MEDIA_FAIL
: Indicates that the modify media operation failed. The characteristics of the media session remain as they were before the function was called.

Cautions

- An application must ensure that the streaming properties at two endpoints are in sync. For example, changing coder properties at one endpoint without synchronization with the other endpoint will cause dropped RTP packets.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
: Invalid parameter; invalid coder; invalid count of media information; T.38 is in session; multicast IP address specified; multicast server or client direction.

EIPM_FWERR
: error in lower-level software

EIPM_INTERNAL
: Internal error

EIPM_INV_MODE
: Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made. Stream is idle.

EIPM_SYSTEM
System error

The **ipm_ModifyMedia( )** function returns security errors in the following circumstances:

1. The total number of keys has exceeded the given limit (IPM_SECURITY_MAX_TOTAL_NUM_KEYS, which is set to 20).

2. The number of keys for any given media type exceeds the given limit (IPM_SECURITY_MAX_NUM_KEYS, which is set to 10).

3. The size of a generated key does not match the key size (master key or master salt) in the specified crypto suite (see Table 6, "Crypto Suite Parameter Values", on page 171).

4. The Secure RTP feature is not available.

5. All Secure RTP resources are allocated.

6. Any of the structure version fields are invalid.

Example

The following sample code changes the coder from G.711 mu-law to G.711 A-law and also changes the direction.

```
#include <stdio.h>
#include <string>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
   /*
   .
   .
   Main Processing
   .
   .
   */

   /*
   Set the media properties for a remote party using IP device handle, nDeviceHandle.
   ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
   */

   IPM_MEDIA_INFO MediaInfo;
   MediaInfo.unCount = 4;

   MediaInfo.MediaData[0].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTP_INFO;
   MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
   strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.9");
```

```
MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
MediaInfo.MediaData[1].mediaInfo.PortInfo.unPortId = 2329;
strcpy(MediaInfo.MediaData[1].mediaInfo.PortInfo.cIPAddress, "111.41.0.9");

MediaInfo.MediaData[2].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unRedPayloadType = 0;

MediaInfo.MediaData[3].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unRedPayloadType = 0;

if (ipm_StartMedia(nDeviceHandle, &MediaInfo, DATA_IP_TDM_BIDIRECTIONAL, EV_SYNC) == -1)
{
   printf("ipm_StartMediaInfo failed for device name = %s with error = %d\n",
   ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
   /*
   .
   .
   Perform Error Processing
   .
   .*/
}
/*
.
. Continue processing
.
*/

MediaInfo.unCount = 2;
MediaInfo.MediaData[0].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ALAW64K;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[0].mediaInfo.CoderInfo.unRedPayloadType = 0;

MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ALAW64K;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[1].mediaInfo.CoderInfo.unRedPayloadType = 0;


if (ipm_ModifyMedia(nDeviceHandle, &MediaInfo, DATA_IP_SENDONLY, EV_SYNC) == -1)
{
   printf("ipm_Modify failed for device name = %s with error = %d\n",
   ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
   /*
   .
   .
   Perform Error Processing
   */
}
```

```
    /*
    .
    .
    continue processing
    .
    */

}
```

See Also

- ipm_StartMedia( )

ipm_Open( )

Name: int ipm_Open(*szDevName, *pOpenInfo, usMode)

Inputs:

const char *szDeviceName	• device name pointer
IPM_OPEN_INFO *pOpenInfo	• set to NULL
unsigned short usMode	• async or sync mode setting

Returns: device handle if successful
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_Open( )** function opens an IP channel or board device and returns a unique device handle to identify the physical device that performs the media transfer. All subsequent references to the opened device must be made using the handle until the device is closed.

The IP Media library allows **ipm_Open( )** to be called either synchronously or asynchronously.

If **ipm_Open( )** is called synchronously and no errors are received, the device handle that is returned is valid and may be used by the application.

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

If **ipm_Open( )** is called asynchronously and IPMEV_ERROR is returned, a device handle is also returned. The application must call **ipm_Close( )** using the handle returned by **ipm_Open( )**.

Parameter	Description
szDeviceName	pointer to device name to open
	IP Media channel device: **ipmBxCy** where **x** is the unique logical board number and **y** is the media device channel number.
	Board device: **ipmBx** where **x** is the unique logical board number.
pOpenInfo	set to NULL; reserved for future use
usMode	operation mode
	Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_OPEN
Indicates successful completion; that is, an IP channel was opened and the device handle is valid. This event does not return any data.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- Two different applications (running in separate processes) cannot use the same IP media device (ipmBxCx). In other words, multiple calls to **ipm_Open( )** on the same IP media device are not allowed.
- The **pOpenInfo** pointer is reserved for future use and must be set to NULL.
- If this function is called asynchronously and IPMEV_ERROR is received, the application must call **ipm_Close( )** using the handle returned by **ipm_Open( )**.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EINVAL
Invalid argument (system-level error)

ENOMEM
Memory allocation failure (system-level error)

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    char cDevName[10];
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    . Create a Thread that waits on srl events, this
    . thread will execute the WorkerThread function
    .
    .
    */

    /*
    Open IP channel ipmB1C1
    */
    sprintf(cDevName,"ipmB1C%d", 1);
        if((nDeviceHandle = ipm_Open(cDevName, NULL, EV_ASYNC)) == -1)
```

```
        {
            printf("ipm_Open failed for device name = %s\n", cDevName);
            /*
            .
            .
            Perform Error Processing
            .
            .
            */
        }
    /*
    .
    .
    . continue Main Processing
    .
    .
    */
}

void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_Open */
        case IPMEV_OPEN:
            printf("Received IPMEV_OPEN for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_Close( )**

ipm_ReceiveDigits( )

Name: int ipm_ReceiveDigits(nDeviceHandle, *pDigitInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_DIGIT_INFO *pDigitInfo	• pointer to digit info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_ReceiveDigits( )** function enables the IP channel to receive digits from the TDM bus. The receive operation continues until **ipm_Stop( )** is called with the eSTOP_RECEIVE_DIGITS flag set.

Note that digits are *always* received asynchronously, even though this function may be called in either asynchronous or synchronous mode. If this function is called synchronously and returns 0, it does not indicate that the digits have been received; instead, it only indicates that the function was successfully processed by the firmware. The application must enable event reporting and check for IPMEV_DIGITS_RECEIVED events.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pDigitInfo	pointer to data structure that contains digit information. The application must set the direction and type of digits before calling the function. On return, the function sets the unNumberOfDigits field to indicate how many IPMEV_DIGITS_RECEIVED events the application must process. See IPM_DIGIT_INFO for details. **Note**: The cDigits[MAX_IP_DIGITS] field in the data structure pointed to by **pDigitInfo** is not used by the **ipm_ReceiveDigits( )** function in either SYNC or ASYNC mode. The incoming digit is delivered asynchronously in the IPM_DIGIT_INFO data structure associated with an IPMEV_DIGITS_RECEIVED event.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_RECEIVE_DIGITS
Indicates function was successfully processed but does **not** indicate that digits were received. This event does not return data.

IPMEV_ERROR
Indicates that the function failed.

Note: IPMEV_DIGITS_RECEIVED is an unsolicited event that may be reported after the **ipm_ReceiveDigits( )** function is called either synchronously or asynchronously. An event is reported for each digit that was received. The event data indicates the digit origin via the eIPM_DIGIT_DIRECTION enumeration.

Cautions

- The only supported value for IPM_DIGIT_INFO.eIPM_DIGIT_DIRECTION is to receive digits from the TDM bus.
- The **ipm_ReceiveDigits( )** function returns valid data only if the digits are being transmitted in out-of-band mode. For more information on setting DTMF mode, see the *Dialogic® IP Media Library API Programming Guide*.
- Digits are only received if there is an active RTP session; if two IPM devices are directly routed together, you must establish an RTP session before digits can be sent and received.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <ipmlib.h>
#include <srllib.h>
#include <stdio.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    IPM_DIGIT_INFO myDigitInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
```

```
    /*
    .
    .
     Main Processing
    .
    .
    */

    /*
     Enable an IP device handle, nDeviceHandle, to receive a specified set of digits.
     ASSUMPTION:  A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */

    myDigitInfo.eDigitType = DIGIT_ALPHA_NUMERIC;
    myDigitInfo.eDigitDirection = DIGIT_TDM;

    if(ipm_ReceiveDigits(nDeviceHandle, &myDigitInfo, EV_ASYNC) == -1)
    {
        printf("ipm_ReceiveDigits failed for device name = %s with error = %d\n",
              ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
     Continue processing
    .
    .
    */
}

void CheckEvent()
{

    IPM_DIGIT_INFO *pDigitInfo;
    int nDeviceID  = sr_getevtdev();
    int nEventType = sr_getevttype();
    void* pVoid    = sr_getevtdatap();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        //Successful reply to ipm_ReceiveDigits()
        case IPMEV_RECEIVE_DIGITS:
            printf("Received IPMEV_RECEIVE_DIGITS for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;
```

```
        //Unsolicited event, retrieve digits
        case IPMEV_DIGITS_RECEIVED:
            printf("Received IPM_DIGITS_RECEIVED for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            pDigitInfo = (IPM_DIGIT_INFO*)pVoid;
            printf("Number of digits = %d, digit=%s on device %s\n",
                pDigitInfo->unNumberOfDigits, pDigitInfo->cDigits,
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_SendDigits( )**

ipm_ResetQoSAlarmStatus( )

Name: int ipm_ResetQoSAlarmStatus(nDeviceHandle, *pQoSAlarmInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_QOS_ALARM_STATUS *pQoSAlarmInfo	• pointer to QoS alarm structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software

Description

The **ipm_ResetQoSAlarmStatus( )** function resets to the OFF state one or more Quality of Service (QoS) alarms that report the status of a media channel.This function does not apply to board-level alarms.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pQoSAlarmInfo	pointer to IPM_QOS_ALARM_STATUS structure which contains one or more IPM_QOS_ALARM_DATA structures
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_RESET_QOS_ALARM_STATUS
Indicates successful completion; that is, specified QoS alarm(s) has been reset to OFF. This event does not return data.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    IPM_QOS_ALARM_STATUS myAlarmStatus;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Reset the QOSTYPE_JITTER alarm for IP device handle, nDeviceHandle.
    NOTE: nDeviceHandle was obtained from prior call to ipm_Open()
    */
    myAlarmStatus.unAlarmCount = 1;
    myAlarmStatus.QoSData[0].eQoSType = QOSTYPE_JITTER;
    if(ipm_ResetQoSAlarmStatus(nDeviceHandle, &myAlarmStatus, EV_ASYNC) == -1)
    {
        printf("ipm_ResetQoSAlarmStatus failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
         Perform Error Processing
        .
        .
        */
    }
```

```
    /*
    .
    .
    Continue Processing
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        Other events
        .
        .
        */

        /* Expected reply to ipm_ResetQoSAlarmStatus */
        case IPMEV_RESET_QOS_ALARM_STATUS:
            printf("Received IPMEV_RESET_QOS_ALARM_STATUS for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_GetQoSAlarmStatus( )**

ipm_SecurityGenMasterKeys( )

Name: int ipm_SecurityGenMasterKeys(nDeviceHandle, *pKeys, usNumKeys, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_SECURITY_KEY *pKeys	• pointer to security key structure
unsigned short usNumKeys	• number of keys to be generated
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: synchronous

Platform: Dialogic® HMP Software

Description

The **ipm_SecurityGenMasterKeys( )** function generates master and salt keys. The security keys can be generated either in binary format or Base64-encoded format. This function can be called any time after opening an IP Media device. The function is supported in synchronous mode only.

Parameter	Description
nDeviceHandle	handle of the IP Media device
***pKeys**	A pointer to an array of IPM_SECURITY_KEY structures. Applications need to allocate the memory for each IPM_SECURITY_KEY structure and fill in the version of the structures, type of the key that needs to be generated, pointers to the corresponding key structure, master key length, and master salt key length. This function does not modify any other fields of the structure except the keys generated.
usNumKeys	indicates the number of keys that need to be generated
usMode	operation mode, which must be set to EV_SYNC for synchronous execution

Termination Events

None.

Cautions

The application is expected to perform all memory allocation for the requested keys, that includes the array of IPM_SECURITY_KEY structures as well as for the individual keys pointed to by the pvMasterKey field in each IPM_SECURITY_KEY structure.

Errors

The function returns an error if:

1. Any of the structure version fields are invalid.

2. The pKeys field in the IPM_SECURITY_INFO structure or the pvMasterKey field in the IPM_SECURITY_KEY structure is NULL.

3. The pcMasterKey or pcMasterSaltKey fields in the IPM_SECURITY_BINARY_KEY structure or the pcMasterBase64Key field in the IPM_SECURITY_BASE64_KEY structure is NULL.

4. The key lengths are not valid.

5. The device handle is not valid.

6. The mode is not EV_SYNC.

Example

```
/*the following sample code uses SRTP */

#include <stdio.h>
#include <string>
#include <srllib.h>
#include <ipmlib.h>
typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    /*
    .
    .
    Main Processing
    .
    */

    /*
    Set the keys for the IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */

    char Mkey1[30], bs64Mkey1[40];
    char Msalt1[30];

    IPM_SRTP_PARMS srtp_parms;
    IPM_SECURITY_KEY MasterKeys[2];
    IPM_SECURITY_BASE64_KEY Masterbs64Key;
    IPM_SECURITY_BINARY_KEY MasterbinKey;

    INIT_IPM_SRTP_PARMS(&srtp_parms);
    INIT_IPM_SECURITY_BINARY_KEY(&MasterbinKey);

    MasterbinKey.pcMasterKey = Mkey1;
    MasterbinKey.pcMasterSaltKey = Msalt1;

    INIT_IPM_SECURITY_BASE64_KEY(&Masterbs64Key);
    Masterbs64Key.pcMasterBase64Key = bs64Mkey1;

    INIT_IPM_SECURITY_KEY(&MasterKeys[0]);
```

```
MasterKeys[0].eKeyType = IPM_SECURITY_KEYTYPE_BINARY;
MasterKeys[0].pvMasterKey = &MasterbinKey;

INIT_IPM_SECURITY_KEY(&MasterKeys[1]);
MasterKeys[1].eKeyType = IPM_SECURITY_KEYTYPE_BASE64;
MasterKeys[1].pvMasterKey = &Masterbs64Key;


/* Generate the master Key and Master Salt Key for the device */

if ((ipm_SecurityGenMasterKeys(nDeviceHandle, MasterKeys, 1, EV_SYNC) == -1))
{
    printf("ipm_SecurityGenMasterKeys() failed for device name = %s with error = %d\n",
    ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
    /*
    .
    .
    Perform Error Processing
    */
}


/* Masterbs64Key can be filled from SDP */



/*
Set the media properties for a remote party using IP device handle, nDeviceHandle.
ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
*/

IPM_MEDIA_INFO MediaInfo;
MediaInfo.unCount = 6;


MediaInfo.MediaData[0].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTP_INFO;
MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.9\n");

MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
MediaInfo.MediaData[1].mediaInfo.PortInfo.unPortId = 2329;
strcpy(MediaInfo.MediaData[1].mediaInfo.PortInfo.cIPAddress, "111.41.0.9\n");

MediaInfo.MediaData[2].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.unRedPayloadType = 0;

MediaInfo.MediaData[3].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[3].mediaInfo.CoderInfo.unRedPayloadType = 0;


MediaInfo.MediaData[4].eMediaType = MEDIATYPE_AUDIO_REMOTE_SECURITY_INFO;
MediaInfo.MediaData[4].mediaInfo.SecurityInfo.unVersion = IPM_SECURITY_INFO_VERSION;
MediaInfo.MediaData[4].mediaInfo. SecurityInfo.unNumKeys = 1;
MediaInfo.MediaData[4].mediaInfo. SecurityInfo.pParms = &srtp_parms;
MediaInfo.MediaData[4].mediaInfo. SecurityInfo.pKeys = &MasterKeys[0];
MediaInfo.MediaData[4].mediaInfo. SecurityInfo.eInfoMode = IPM_SECURITY_INFO_MODE_IMMEDIATE;
```

```
    MediaInfo.MediaData[5].eMediaType = MEDIATYPE_AUDIO_LOCAL_SECURITY_INFO;
    MediaInfo.MediaData[5].mediaInfo. SecurityInfo.unVersion = IPM_SECURITY_INFO_VERSION;
    MediaInfo.MediaData[5].mediaInfo. SecurityInfo.unNumKeys = 1;
    MediaInfo.MediaData[5].mediaInfo. SecurityInfo.pParms = &srtp_parms;
    MediaInfo.MediaData[5].mediaInfo. SecurityInfo.pKeys = &MasterKeys[1];
    MediaInfo.MediaData[5].mediaInfo. SecurityInfo.eInfoMode = IPM_SECURITY_INFO_MODE_IMMEDIATE;


    if (ipm_StartMedia(nDeviceHandle, &MediaInfo, DATA_IP_TDM_BIDIRECTIONAL,EV_SYNC) == -1)
    {
        printf("ipm_StartMediaInfo failed for device name = %s with error = %d\n",
        ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        */
    }
    /*
    .
    . Continue processing
    .
    */


}
```

See Also

- **ipm_StartMedia( )**
- **ipm_ModifyMedia( )**

ipm_SendDigits( )

Name: int ipm_SendDigits(nDeviceHandle, *pDigitInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_DIGIT_INFO *pDigitInfo	• pointer to digit info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software

Description

The **ipm_SendDigits**( ) function generates the supplied digits to the TDM bus.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pDigitInfo	pointer to structure that contains digit type, direction, and digits; see IPM_DIGIT_INFO for details. Note that the application must fill in the digit type, direction, number of digits, and the actual digits to be sent. The maximum number of digits is 16.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_SEND_DIGITS
Indicates successful completion; that is, the supplied digits were sent. This event does not return data.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- If this function is called synchronously and returns 0, it does not indicate that the digits have been sent, but only that the function was successfully processed by the firmware. The application must enable event reporting and check for the IPMEV_SEND_DIGITS event.

- The only supported value for IPM_DIGIT_INFO.eIPM_DIGIT_DIRECTION is to send digits toward the TDM bus.
- Digits are only exchanged if there is an active RTP session; if two IPM devices are directly routed together, you must establish an RTP session before digits can be sent and received.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <string.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    IPM_DIGIT_INFO myDigitInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Generate a set of digits using IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    myDigitInfo.eDigitType = DIGIT_ALPHA_NUMERIC;
    myDigitInfo.eDigitDirection = DIGIT_TDM;
    strcpy(myDigitInfo.cDigits,"1234567890123456");
    myDigitInfo.unNumberOfDigits = 16;
    if(ipm_SendDigits(nDeviceHandle, &myDigitInfo, EV_ASYNC) == -1)
    {
        printf("ipm_SendDigits failed for device name = %s with error = %d\n",
```

```
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    . Continue Main processing
    .
    .
    */
}

void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    void* pVoid = sr_getevtdatap();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        //Successful reply to ipm_SendDigits()
        case IPMEV_SEND_DIGITS:
            printf("Received IPMEV_SEND_DIGITS for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_ReceiveDigits( )**

ipm_SetParm( )

Name: int ipm_SetParm(nDeviceHandle, *pParmInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_PARM_INFO *pParmInfo	• pointer to parameter info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_SetParm**( ) function sets values for the specified parameter.

Parameter	Description
nDeviceHandle	handle of the IP media device
pParmInfo	pointer to structure that contains IP channel parameter values See the IPM_PARM_INFO data structure page for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_SET_PARM
Indicates successful completion; that is, the supplied IP channel parameter was modified.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR**( ) and **ATDV_ERRMSGP**( ) to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    ASSUMPTION: A valid nDeviceHandle was obtained from prior
    call to ipm_Open().
    */
    IPM_PARM_INFO ParmInfo;
    unsigned long ulParmValue = ECHO_TAIL_16;
    ParmInfo.eParm = PARMCH_ECHOTAIL;
    ParmInfo.pvParmValue = &ulParmValue;
    if(ipm_SetParm(nDeviceHandle, &ParmInfo, EV_ASYNC)==-1)
    {
        printf("ipm_SetParm failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    continue
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
```

```
    switch(nEventType)
    {
        /*
        .
        .
        Other events
        .
        .
        */

        /* Expected reply to ipm_GetQoSAlarmStatus */
        case IPMEV_SET_PARM:
            printf("Received IPMEV_SETPARM for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_GetParm( )**

ipm_SetQoSThreshold( )

Name: int ipm_SetQoSThreshold(nDeviceHandle, *pQoSThresholdInfo, usMode)

Inputs:

int nDeviceHandle	• IP Media channel device handle
IPM_QOS_THRESHOLD_INFO *pQoSThresholdInfo	• pointer to QoS alarm threshold structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

■ Description

The **ipm_SetQoSThreshold( )** function changes alarm threshold settings for Quality of Service (QoS) alarms that report the status of a media channel.

This function can be called at any time, including when a session is in progress.

Parameter	Description
nDeviceHandle	handle of the IP Media channel device
pQoSThresholdInfo	pointer to IPM_QOS_THRESHOLD_INFO structure which contains one or more IPM_QOS_THRESHOLD_DATA structures with the threshold settings to be set. Note that when an application needs to specify any given field in an IPM_QOS_THRESHOLD_DATA structure, it must populate *all* fields in the structure even if those fields are to remain at their default values.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

■ Termination Events

IPMEV_SET_QOS_THRESHOLD_INFO
Indicates successful completion; that is, alarm QoS threshold levels were modified. Use SRL functions to retrieve IPM_QOS_THRESHOLD_INFO structure fields.

IPMEV_ERROR
Indicates that the function failed.

Cautions

If an application exits without calling **ipm_UnListen( )** to clean up voice device routings, the **ipm_SetQoSThreshold( )** function may fail if it is called after the application is restarted but before a new routing of the IPM device to a voice device is established.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

void CheckEvent();
typedef long int(*HDLR)(unsigned long);

void main()
{
    int nDeviceHandle;
    IPM_QOS_THRESHOLD_INFO mySetQosThresholdInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Change two alarm threshold settings for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    mySetQosThresholdInfo.unCount = 2;
    mySetQosThresholdInfo.QoSThresholdData[0].eQoSType = QOSTYPE_LOSTPACKETS;
    mySetQosThresholdInfo.QoSThresholdData[0].unTimeInterval = 100;
    mySetQosThresholdInfo.QoSThresholdData[0].unDebounceOn = 100;
    mySetQosThresholdInfo.QoSThresholdData[0].unDebounceOff = 100;
    mySetQosThresholdInfo.QoSThresholdData[0].unFaultThreshold = 20;
    mySetQosThresholdInfo.QoSThresholdData[0].unPercentSuccessThreshold = 60;
```

```
    mySetQosThresholdInfo.QoSThresholdData[0].unPercentFailThreshold = 40;
    mySetQosThresholdInfo.QoSThresholdData[1].eQoSType = QOSTYPE_JITTER;
    mySetQosThresholdInfo.QoSThresholdData[1].unTimeInterval = 100;
    mySetQosThresholdInfo.QoSThresholdData[1].unDebounceOn = 200;
    mySetQosThresholdInfo.QoSThresholdData[1].unDebounceOff = 600;
    mySetQosThresholdInfo.QoSThresholdData[1].unFaultThreshold = 60;
    mySetQosThresholdInfo.QoSThresholdData[1].unPercentSuccessThreshold = 60;
    mySetQosThresholdInfo.QoSThresholdData[1].unPercentFailThreshold = 40;

    if(ipm_SetQoSThreshold(nDeviceHandle, &mySetQosThresholdInfo, EV_ASYNC) == -1)
    {
        printf("ipm_SetQoSThreshold failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

    /*
    .
    .
    . continue
    .
    .
    */
}

void CheckEvent()
{
    //Get event type and associated data
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */

        /* Expected reply to ipm_SetQoSThreshold */
        case IPMEV_SET_QOS_THRESHOLD_INFO:
            printf("Received IPMEV_SET_QOS_THRESHOLD_INFO for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_GetQoSThreshold( )**

ipm_StartMedia( )

Name: int ipm_StartMedia(nDeviceHandle, *pMediaInfo, eDirection, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
IPM_MEDIA_INFO *pMediaInfo	• pointer to media information structure
eIPM_DATA_DIRECTION eDirection	• data flow direction
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_StartMedia( )** function sets media properties and starts the session. This function allows the application to set the remote and local connectivity selections. **ipm_StartMedia( )** also starts RTP streaming. The remote RTP/ RTCP port information and coder information is provided in the IPM_MEDIA_INFO structure.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pMediaInfo	media information data structure; see IPM_MEDIA_INFO for details Applications can define the following: • local transmit coder and remote transmit coder • local and remote RTP/RTCP port • local and remote IP address • local and remote T.38 port

Parameter	Description
eDirection	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: • DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data • DATA_IP_SENDONLY – send data to the IP network but do not receive data • DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_STARTMEDIA
Indicates successful completion; that is, media information was set and the session has been started.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- For an asynchronous call, it is recommended that the **ipm_Listen( )** function should be called after the **ipm_StartMedia( )** completion event has been delivered.
- Do not set the IP address to 0.0.0.0, because this may lead to a hung port.
- For the Dialogic® Multimedia Platform for AdvancedTCA, this function returns an invalid state error if a coder is not reserved prior to calling **ipm_GetLocalMediaInfo( )**. All audio and/or video coders must be reserved using the **dev_ReserveResourceEx( )** Dialogic® Device Management Library function. However, this restriction does not apply when using a fax (T.38) media type.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_BUSY
Channel is busy

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

The **ipm_StartMedia( )** function returns security errors in the following circumstances:

- The total number of keys has exceeded the given limit (IPM_SECURITY_MAX_TOTAL_NUM_KEYS, which is set to 20).
- The number of keys for any given media type exceeds the given limit (IPM_SECURITY_MAX_NUM_KEYS, which is set to 10).
- The size of a generated key does not match the key size (master key or master salt) in the specified crypto suite (see Table 6, "Crypto Suite Parameter Values", on page 171).
- The Secure RTP feature is not available.
- All Secure RTP resources are allocated.
- Any of the structure version fields are invalid.

Example

```
#include <stdio.h>
#include <string>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
     int nDeviceHandle, nMediaCnt = 0;
     // Register event handler function with srl
     IPM_MEDIA_INFO MediaInfo;

     sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
     /*
     .
     .
     Main Processing
     .
     .
     */
     /*
     Set the media properties for a remote party using IP device handle, nDeviceHandle.
     ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
     */

     /* clear the RemoteMediaInfo structures */
     memset(&MediaInfo, 0, sizeof(IPM_MEDIA_INFO));

     /* remote audio */
     MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eFrameSize =
               (eIPM_CODER_FRAMESIZE)m_nAudioFrameSize;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unFramesPerPkt = 1;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unCoderPayloadType = 0;
     MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unRedPayloadType = 0;

     /* local audio */
```

```
++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eFrameSize =
        (eIPM_CODER_FRAMESIZE)m_nAudioFrameSize;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unFramesPerPkt = 1;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unCoderPayloadType = 0;
MediaInfo.MediaData[nMediaCnt].mediaInfo.CoderInfo.unRedPayloadType = 0;

/* remote video */
++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_VIDEO_REMOTE_CODER_INFO;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.eCoderType = CODER_TYPE_H263;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.unVersion = 0;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.unCoderPayloadType = 100;

/* local video coder */
++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_VIDEO_LOCAL_CODER_INFO;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.eCoderType = CODER_TYPE_H263;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.unVersion = 0;
MediaInfo.MediaData[nMediaCnt].mediaInfo.VideoCoderInfo.unCoderPayloadType = 100;

/* remote audio port */
++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTP_INFO;
strcpy(MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.cIPAddress,"192.168.0.9");
MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.unPortId = 2328;

++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
strcpy(MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.cIPAddress,"192.168.0.9");
MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.unPortId = 2329;

/* remote video port */
++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_VIDEO_REMOTE_RTP_INFO;
strcpy(MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.cIPAddress,"192.168.0.9");
MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.unPortId = 2340;

++nMediaCnt;
MediaInfo.MediaData[nMediaCnt].eMediaType = MEDIATYPE_VIDEO_REMOTE_RTCP_INFO;
strcpy(MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.cIPAddress,"192.168.0.9");
MediaInfo.MediaData[nMediaCnt].mediaInfo.PortInfo.unPortId = 2341;

MediaInfo.unCount = nMediaCnt + 1;

if(ipm_StartMedia(nDeviceHandle, &MediaInfo, DATA_IP_TDM_BIDIRECTIONAL, EV_ASYNC) == -1)
{
     printf("ipm_StartMediaInfo failed for device name = %s with error = %d\n",
               ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
     /*
     .
     Perform Error Processing
     .
     .
     */
}

/*
.
.
Continue processing
.
.
*/
```

```
}

void CheckEvent()
{
     int nDeviceID = sr_getevtdev();
     int nEventType = sr_getevttype();
     switch(nEventType)
     {
          /*
          .
          .
          Other events
          .
          .
          */
          /* Expected reply to ipm_StartMedia */
          case IPMEV_STARTMEDIA:
               printf("Received IPMEV_STARTMEDIA for device = %s\n", ATDV_NAMEP(nDeviceID));
               break;

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

See Also

- **ipm_ModifyMedia( )**
- **ipm_Stop( )**

ipm_Stop( )

Name: int ipm_Stop(nDeviceHandle, eOperation, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
eIPM_STOP_OPERATION eOperation	• operation to be stopped
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_Stop( )** function stops operations on the specified IP channel.

To run this function asynchronously, set **mode** to EV_ASYNC. The function returns 0 if successful and the application must wait for the IPMEV_STOPPED event.

Parameter	Description
nDeviceHandle	handle of the IP Media device
eOperation	the type of operation(s) to stop; only one value can be set at a time The eIPM_STOP_OPERATION data type is an enumeration that defines the following values: • STOP_RECEIVE_DIGITS – stop receiving digits • STOP_MEDIA – operation of media streaming session. This enumeration disconnects the session. The application must call **ipm_StartMedia( )** to start a new session. • STOP_ALL – stop all operations
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_STOPPED
Indicates that activity of the type specified in **eOperation** has terminated on this channel. This event does not return data.

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Application needs to stop a current session on IP device handle, nDeviceHandle
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open()
    and a session has been started by calling ipm_StartMedia() some time earlier.
    */
    if(ipm_Stop(nDeviceHandle, STOP_ALL, EV_ASYNC) == -1)
    {
        printf("ipm_Stop failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
```

```
    /*
    .
    .
    Continue Processing
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        List of expected events
        .
        .
        */

        /* Expected reply from ipm_Stop() */
        case IPMEV_STOPPED:
            printf("Received IPMEV_STOPPED for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- **ipm_UnListen( )**

ipm_UnListen( )

Name: int ipm_UnListen(nDeviceHandle, usMode)

Inputs: int nDeviceHandle • IP Media device handle

unsigned short usMode • async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: Dialogic® HMP Software, Dialogic® Multimedia Platform for AdvancedTCA

Description

The **ipm_UnListen( )** function stops listening to the TDM time slot specified in a previous call to **ipm_Listen( )**.

If **ipm_Listen( )** is called to connect to a different TDM time slot, the firmware automatically breaks an existing connection and reconnects it to the new time slot. In this case, the application does not need to call the **ipm_UnListen( )** function.

Parameter	Description
nDeviceHandle	handle of the IP Media device
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_UNLISTEN
Indicates successful completion; that is, the IP channel was disconnected from the specified TDM time slot. This event does not return data.

IPMEV_ERROR
Indicates that the function failed.

Cautions

- The IP Media library allows **ipm_Listen( )** and **ipm_UnListen( )** to be called either synchronously or asynchronously. Other Dialogic libraries may not support asynchronous execution of the similar xx_Listen and xx_UnListen functions.

Errors

If the function returns -1 to indicate failure, call **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to return one of the following errors:

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

EIPM_INV_STATE
Invalid state. Initial command did not complete before another function call was made.

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <srllib.h>
#include <ipmlib.h>

typedef long int(*HDLR)(unsigned long);
void CheckEvent();

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

    /*
    Stop an IP device handle, nDeviceHandle, from listening to a time slot.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_UnListen(nDeviceHandle, EV_ASYNC) == -1)
    {
        printf("ipm_UnListen failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
```

```
    /*
    .
    .
    Continue processing
    .
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();

    switch(nEventType)
    {
        /*
        .
        .
        Other events
        .
        .
        */

        /*Expected reply from ipm_UnListen*/
        case IPMEV_UNLISTEN:
            printf("Received IPMEV_UNLISTEN for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- ipm_Listen( )
- ipm_Stop( )

Events 3

This chapter describes the events that are returned by the Dialogic® IP Media Library API functions. The function descriptions in Chapter 2, "Function Information" lists the function's termination events for asynchronous operations.

There are three types of events returned by the Dialogic® IP Media Library API functions:

- events returned after the termination of a function call, called termination events
- unsolicited events triggered by external events
- notification events requested (solicited) by the application

Applications can enable or disable certain notification events for Quality of Service (QoS) information. The notification events supported by the Dialogic® IP Media Library API are enabled and disabled via the function calls **ipm_EnableEvents( )** and **ipm_DisableEvents( )**, respectively.

The following events, listed in alphabetical order, may be returned by the Dialogic® IP Media Library API software. Use **sr_waitevt( )**, **sr_enbhdlr( )** or other SRL functions to collect an event code, depending on the programming model in use. For more information, see the *Dialogic® Standard Runtime Library API Library Reference*.

IPMEV_DIGITS_RECEIVED
: Unsolicited event for **ipm_ReceiveDigits( )** in either synchronous or asynchronous mode. One event is returned for each digit that is received. Event contains digit data in IPM_DIGIT_INFO data structure.

IPMEV_ERROR
: Generic unsuccessful termination event. This event may be generated on any handle when there is an error. No data is returned in the event.

IPMEV_EVENT_DISABLED
: Successful termination event for **ipm_DisableEvents( )**. Indicates that IP notification events specified in function call have been disabled. No data is returned in the event.

IPMEV_EVENT_ENABLED
: Successful termination event for **ipm_EnableEvents( )**. Indicates that IP notification events specified in the function call have been enabled. No data is returned in the event.

IPMEV_FAXTONE
: Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when fax tone is detected on TDM. Contains fax tone information in an IPM_FAX_SIGNAL data structure.

IPMEV_GET_LOCAL_MEDIA_INFO
: Successful termination event for **ipm_GetLocalMediaInfo( )**. Contains requested local media information in an IPM_MEDIA_INFO structure.

IPMEV_GET_PARM
: Successful termination event for **ipm_GetParm( )**. Contains requested IP channel parameters in an IPM_PARM_INFO structure.

IPMEV_GET_QOS_ALARM_STATUS
Successful termination event for **ipm_GetQoSAlarmStatus( )**. Contains requested alarm status information in an IPM_QOS_ALARM_STATUS data structure.

IPMEV_GET_QOS_THRESHOLD_INFO
Successful termination event for **ipm_GetQoSThreshold( )**. Contains requested alarm threshold settings in an IPM_QOS_THRESHOLD_INFO data structure.

IPMEV_GET_SESSION_INFO
Successful termination event for **ipm_GetSessionInfo( )**. Contains statistics for previous session in an IPM_SESSION_INFO data structure.

IPMEV_GET_XMITTS_INFO
Successful termination event for **ipm_GetXmitSlot( )**. Contains requested TDM time slot information in an SC_TSINFO data structure.

IPMEV_INIT_COMPLETE
Unsolicited event for **ipm_InitResponseSend( )** and **ipm_InitSend( )** reported when the Nb UP session has been successfully negotiated with the remote party. Both parties may now begin exchanging data. See also IPMEV_INIT_SEND and IPMEV_INIT_RESPONSE_SEND.

IPMEV_INIT_FAILED
Unsolicited event for **ipm_InitResponseSend( )** and **ipm_InitSend( )** reported upon encountering an unexpected error during initialization or a negative response (NACK) from the remote party. Event Data: IPM_INIT_FAILURE structure. See also IPMEV_INIT_SEND and IPMEV_INIT_RESPONSE_SEND.

IPMEV_INIT_RECEIVED
Unsolicited event telling the application that there is an incoming request for an Nb UP session. The application may respond to this message by calling **ipm_InitResponseSend( )**.

IPMEV_INIT_RESPONSE_SEND
Successful termination event for **ipm_InitResponseSend( )**. Indicates successful completion of sending the initialization response message. It does not indicate whether the event was received by the remote party. If IPMEV_INIT_RESPONSE_SEND is received, one of the following unsolicited events is subsequently reported: IPMEV_INIT_COMPLETE or IPMEV_INIT_FAILED.

IPMEV_INIT_SEND
Successful termination event for **ipm_InitSend( )**. Indicates successful completion of sending the initialization message. It does not indicate whether the event was received by remote party or what the response was. If IPMEV_INIT_SEND is received, one of the following unsolicited events is subsequently reported: IPMEV_INIT_COMPLETE or IPMEV_INIT_FAILED.

IPMEV_LISTEN
Successful termination event for **ipm_Listen( )**. Indicates time slot routing was successfully completed. No data is returned in the event.

IPMEV_MODIFY_MEDIA
Successful termination event for **ipm_ModifyMedia( )**. Indicates change of media characteristics was successfully completed. No data is returned in the event.

IPMEV_MODIFY_MEDIA_FAIL
Unsuccessful termination event for **ipm_ModifyMedia( )**. Indicates that the media session was not changed.

IPMEV_NACK_SENT
Unsolicited event telling the application that the IPM library was forced to send a NACK message to the other end of the connection. An example is if a request comes in that the library does not understand. This is different from sending a NACK response to an IPM_INIT_RECEIVED event.

IPMEV_OPEN
Successful termination event for **ipm_Open( )**. Indicates IP channel was successfully opened and device handle is valid. No data is returned in the event.

IPMEV_QOS_ALARM
Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when desired QoS alarm triggers. See the code example (specifically the CheckEvent( ) function definition) in the "Example Code for QoS Alarm Handling" section of the *Dialogic® IP Media Library API for Host Media Processing Programming Guide* for information on determining which alarm triggered the event.

IPMEV_RECEIVE_DIGITS
Successful termination event for **ipm_ReceiveDigits( )**. Indicates channel has been enabled to receive digits. No data is returned in the event.

Note: IPMEV_DIGITS_RECEIVED is used to indicate when digit transfer has occurred.

IPMEV_RESET_QOS_ALARM_STATUS
Successful termination event for **ipm_ResetQoSAlarmStatus( )**. Indicates specified QoS alarms have been reset to OFF state. No data is returned in the event.

IPMEV_SEC_NOTIFY_EXPIRE_KEY_AUDIO
Unsolicited event enabled via **ipm_EnableEvents( )**. Provides notification that the encryption key for the audio media type is about to expire in the predefined time interval. The time interval can be set using the **ipm_SetParm( )** function and the **PARMCH_NOTIFY_EXPIRE_KEY_AUDIO** parameter.

IPMEV_SEC_NOTIFY_EXPIRE_KEY_VIDEO
Unsolicited event enabled via **ipm_EnableEvents( )**. Provides notification that the encryption key for the video media type is about to expire in the predefined time interval. The time interval can be set using the **ipm_SetParm( )** function and the **PARMCH_NOTIFY_EXPIRE_KEY_VIDEO** parameter.

IPMEV_SEND_DIGITS
Successful termination event for **ipm_SendDigits( )**. Indicates supplied digits were sent successfully. No data is returned in the event.

IPMEV_SET_PARM
Successful termination event for **ipm_SetParm( )**. Indicates IP channel parameters have been modified. No data is returned in the event.

IPMEV_SET_QOS_THRESHOLD_INFO
Successful termination event for **ipm_SetQoSThreshold( )**. Indicates requested changes to QoS alarm threshold levels have been made. The updated threshold information is returned in an IPM_QOS_THRESHOLD_INFO data structure.

IPMEV_STARTMEDIA
Successful termination event for **ipm_StartMedia( )**. Indicates media channel information has been set and session has been started. No data is returned in the event.

IPMEV_STOPPED

Successful termination event for **ipm_Stop( )**. Indicates all on-going activity on the IP channel has terminated. No data is returned in the event.

IPMEV_T38CALLSTATE

Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when T.38 call state changes. Event data is an eIPM_T38CALLSTATE enumeration identifying the new call state.

IPMEV_UNLISTEN

Successful termination event for **ipm_UnListen( )**. Indicates IP channel was disconnected from TDM time slot. No data is returned in the event.

Data Structures 4

This chapter alphabetically lists the data structures used by the Dialogic® IP Media library (IPML) functions. These structures are used to control the operation of functions and to return information. In this chapter, the data structure definition is followed by a table providing a detailed description of the fields in the data structure. These fields are listed in the sequence in which they are defined in the data structure.

CT_DEVINFO

```
typedef struct ct_devinfo {
   unsigned long  ct_prodid;       /* product ID */
   unsigned char  ct_devfamily;    /* device family */
   unsigned char  ct_devmode;      /* device mode */
   unsigned char  ct_nettype;      /* network interface */
   unsigned char  ct_busmode;      /* bus architecture */
   unsigned char  ct_busencoding;  /* bus encoding */
   union {
           unsigned char ct_RFU[7];     /* reserved */
           struct {
                 unsigned char ct_prottype;
           } ct_net_devinfo;
   } ct_ext_devinfo;
} CT_DEVINFO;
```

Description

The CT_DEVINFO data structure supplies information about a device. This structure is used by the **ipm_GetCTInfo( )** function. On return from the function, CT_DEVINFO contains the relevant device and device configuration information.

The valid values for each field of the CT_DEVINFO structure are defined in *ctinfo.h*, which is referenced by *ipmlib.h*. The following descriptions indicate only the values that are relevant when using this structure with the Dialogic® IP Media library. Note that this same data structure definition is used in other Dialogic® API libraries where many additional values may be used.

Field Descriptions

The fields of the CT_DEVINFO data structure are described as follows:

ct_prodid
: contains a valid product identification number for the device

ct_devfamily
: specifies the device family; possible values are:
  - CT_DFHMPDM3 – HMP (Host Media Processing) device
  - CT_DFHMPATCA – Multimedia Platform for AdvancedTCA device

ct_devmode
: specifies the device mode; possible values are:
  - CT_DMNETWORK – DM3 network device

ct_nettype
: specifies the type of network interface for the device; possible values are:
  - CT_NTIPT – IP connectivity

ct_busmode
: specifies the bus architecture used to communicate with other devices in the system; possible values are:
  - CT_BMSCBUS – TDM bus architecture

ct_busencoding
describes the PCM encoding used on the bus; possible values are:
- CT_BEULAW – mu-law encoding
- CT_BEALAW – A-law encoding

ct.ext_devinfo.ct_RFU
not used

ct_ext_devinfo.ct_net_devinfo.ct_prottype
not used

Example

See the Example section for **ipm_GetCTInfo( )**.

IPM_CLOSE_INFO

Description

This structure is used by the **ipm_Close( )** function.

Note: This structure is reserved for future use. NULL must be passed.

IPM_AUDIO_CODER_INFO

IPM_CODER_INFO

```
typedef struct ipm_audio_coder_info_tag
{
   eIPM_CODER_TYPE         eCoderType;         /* The coder Type                    */
   eIPM_CODER_FRAMESIZE    eFrameSize;         /* Frame size supported              */
   unsigned int            unFramesPerPkt;     /* No. of Frames per packet          */
   eIPM_CODER_VAD          eVadEnable;         /* VAD enable/disable flag           */
   unsigned int            unCoderPayloadType; /* Type of coder payload supported   */
   unsigned int            unRedPayloadType;   /* Type of Redundancy Payload        */

} IPM_AUDIO_CODER_INFO, *PIPM_AUDIO_CODER_INFO;
typedef IPM_AUDIO_CODER_INFO  IPM_CODER_INFO;
typedef PIPM_AUDIO_CODER_INFO  PIPM_CODER_INFO;
```

Description

This structure contains the coder properties that will be used in an audio IP session. IPM_AUDIO_CODER_INFO is a child of IPM_MEDIA, which is a child of the IPM_MEDIA_INFO structure. IPM_MEDIA_INFO is used by the **ipm_GetLocalMediaInfo( )** and **ipm_StartMedia( )** functions.

Note: The IPM_AUDIO_CODER_INFO data structure was previously defined under the name IPM_CODER_INFO. The *ipmlib.h* header file includes typedefs that ensure backward compatibility for application code that uses the older name for the data structure.

The EVRC and QCELP coders are supported on the Dialogic® Multimedia Platform for AdvancedTCA only.

Field Descriptions

The fields of the IPM_AUDIO_CODER_INFO data structure are described as follows. Refer to Table 2 for coder-specific guidelines for filling in these fields.

eCoderType

type of coder to be used for streaming media operations

The following values are defined:

- CODER_TYPE_AMRNB_4_75K – GSM AMR-NB, 4.75 kbps, CMR type 0
- CODER_TYPE_AMRNB_4_75K_NATIVE – GSM AMR-NB, 4.75 kbps, CMR type 0
- CODER_TYPE_AMRNB_5_15K – GSM AMR-NB, 5.15 kbps, CMR type 1
- CODER_TYPE_AMRNB_5_15K_NATIVE – GSM AMR-NB, 5.15 kbps, CMR type 1
- CODER_TYPE_AMRNB_5_9K – GSM AMR-NB, 5.9 kbps, CMR type 2
- CODER_TYPE_AMRNB_5_9K_NATIVE – GSM AMR-NB, 5.9 kbps, CMR type 2
- CODER_TYPE_AMRNB_6_7K – GSM AMR-NB, 6.7 kbps, CMR type 3
- CODER_TYPE_AMRNB_6_7K_NATIVE – GSM AMR-NB, 6.7 kbps, CMR type 3
- CODER_TYPE_AMRNB_7_4K – GSM AMR-NB, 7.4 kbps, CMR type 4
- CODER_TYPE_AMRNB_7_4K_NATIVE – GSM AMR-NB, 7.4 kbps, CMR type 4
- CODER_TYPE_AMRNB_7_95K – GSM AMR-NB, 7.95 kbps, CMR type 5
- CODER_TYPE_AMRNB_7_95K_NATIVE – GSM AMR-NB, 7.95 kbps, CMR type 5
- CODER_TYPE_AMRNB_10_2K – GSM AMR-NB, 10.2 kbps, CMR type 6
- CODER_TYPE_AMRNB_10_2K_NATIVE – GSM AMR-NB, 10.2 kbps, CMR type 6

- CODER_TYPE_AMRNB_12_2K – GSM AMR-NB, 12.2 kbps, CMR type 7
- CODER_TYPE_AMRNB_12_2K_NATIVE – GSM AMR-NB, 12.2 kbps, CMR type 7
- CODER_TYPE_AMRNB_NONE – Don't care, CMR type 15. Indicates that the software does not care what bit rate it receives; the eCoderType field sets the CMR value in the transmitted packet. This value is not supported for the MEDIATYPE_AUDIO_REMOTE_CODER_INFO media type. This value is only supported for the MEDIATYPE_AUDIO_LOCAL_CODER_INFO media type.
- CODER_TYPE_EVRC – EVRC
- CODER_TYPE_G711ALAW64K – G.711, A-law, 64 kbps
- CODER_TYPE_G711ALAW64K_NATIVE – G.711, A-law, 64 kbps
- CODER_TYPE_G711ULAW64K – G.711, mu-law, 64 kbps
- CODER_TYPE_G711ULAW64K_NATIVE – G.711, mu-law, 64 kbps
- CODER_TYPE_G7231_5_3K – G.723.1, 5.3 kbps
- CODER_TYPE_G7231_5_3K_NATIVE – G.723.1, 5.3 kbps
- CODER_TYPE_G7231_6_3K – G.723.1, 6.3 kbps
- CODER_TYPE_G7231_6_3K_NATIVE – G.723.1, 6.3 kbps
- CODER_TYPE_G726_16K – G.726, 16 kbps, 2 bits/sample
- CODER_TYPE_G726_16K_NATIVE – G.726, 16 kbps, 2 bits/sample
- CODER_TYPE_G726_24K – G.726, 24 kbps, 3 bits/sample
- CODER_TYPE_G726_24K_NATIVE – G.726, 24 kbps, 3 bits/sample
- CODER_TYPE_G726_32K – G.726, 32 kbps, 4 bits/sample
- CODER_TYPE_G726_32K_NATIVE – G.726, 32 kbps, 4 bits/sample
- CODER_TYPE_G726_40K – G.726, 40 kbps, 5 bits/sample
- CODER_TYPE_G726_40K_NATIVE – G.726, 40 kbps, 5 bits/sample
- CODER_TYPE_G729 – G.729
- CODER_TYPE_G729_NATIVE – G.729
- CODER_TYPE_G729ANNEXA – G.729 Annex A
- CODER_TYPE_G729ANNEXA_NATIVE – G.729 Annex A
- CODER_TYPE_G729ANNEXB – G.729 Annex B
- CODER_TYPE_G729ANNEXB_NATIVE – G.729 Annex B
- CODER_TYPE_G729ANNEXAWANNEXB – G.729 Annex A with Annex B
- CODER_TYPE_G729ANNEXAWANNEXB_NATIVE – G.729 Annex A with Annex B
- CODER_TYPE_QCELP_8K – QCELP, 8 kbps
- CODER_TYPE_QCELP_13K – QCELP, 13 kbps

Note: An IPM device using a coder type that includes _NATIVE as part of the value name for the audio coder is not capable of transcoding audio and can therefore only be used in support of native features such as native play and record operations and native hairpinning. An IPM device using a coder type that does **not** include _NATIVE as part of the value name for the audio coder is capable of transcoding audio and also supports native features.

eFrameSize

size of frame for coders that support multiple frame sizes, such as G.711 and G.726. (Other coders have a predefined, standard value for the frame size and have a user-programmable frames per packet field in the IPM_CODER_INFO data structure.) When packets are sent in both directions (that is, when the call to **ipm_StartMedia( )** or **ipm_SetRemoteMediaInfo( )** specifies **eDirection** = DATA_IP_TDM_BIDIRECTIONAL), the application must know the frame size of incoming packets and use eFrameSize to specify that value.

The eIPM_CODER_FRAMESIZE data type is an enumeration which defines the following values, and coder-specific values are listed in Table 2:

- CODER_FRAMESIZE_10 – frame size = 10 ms
- CODER_FRAMESIZE_20 – frame size = 20 ms
- CODER_FRAMESIZE_30 – frame size = 30 ms

unFramesPerPkt

number of frames per packet. Coder-specific values are listed in Table 2. This field cannot be modified for G.711 coders.

eVadEnable

flag for enabling/disabling VAD (Voice Activity Detection)

The eIPM_CODER_VAD data type is an enumeration which defines the following values, and coder-specific values are listed in Table 2:

- CODER_VAD_DISABLE – VAD is OFF
- CODER_VAD_ENABLE – VAD is ON

unCoderPayloadType

RTP header payload type using RFC 3551 (supercedes RFC1890) standard definitions. The application is responsible for negotiating this value between the two endpoints. This may be set to any value for non-standard coders or if the application does not require interoperability with third-party applications. Values: 0 to 127. 96 to 127 is the dynamic range.

Note: Applications must set a value that is compatible with the coder type that is specified in the eCoderType field before calling **ipm_StartMedia( )** or **ipm_ModifyMedia( )**. If the application does not set this field, the default value of 0 specifies G.711.

unRedPayloadType

RTP header redundancy payload type using RFC 2198 definitions for redundant packets. The application is responsible for negotiating this value between the two endpoints. This may be set to any value from 96 to 127.

Table 2. Supported Audio Coder Properties

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_AMRNB_4_75k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_5_15k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_5_9k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_6_7k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_7_4k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_7_95k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE

NOTES:
HMP = Dialogic® HMP Software; MMP = Dialogic® Multimedia Platform for AdvancedTCA
† On Dialogic® HMP Software (HMP), using fpp value of 1 is not recommended because it requires increased MIPS usage.
‡ Applications must explicitly enable VAD even though G.729a+b implicitly supports VAD
* Supported on Dialogic® Multimedia Platform for AdvancedTCA only (MMP).

Table 2. Supported Audio Coder Properties (Continued)

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_AMRNB_10_2k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_12_2k	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_AMRNB_NONE	fixed at 20	HMP: 1 MMP: 1, 2, 3, 4	Must be CODER_VAD_ENABLE
CODER_TYPE_EVRC *	fixed at 20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ALAW64K	10, 20, or 30	1	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ULAW64K	10, 20, or 30	1	Must be CODER_VAD_DISABLE
CODER_TYPE_G723_1_5_3K	30	HMP: 1, 2, 3 MMP: 1, 2	Either value
CODER_TYPE_G723_1_6_3K	30	HMP: 1, 2, 3 MMP: 1, 2	Either value
CODER_TYPE_G726_16K	20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G726_24K	20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G726_32K	20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G726_40K	20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G729ANNEXA	10	1†, 2, 3, or 4	Must be CODER_VAD_DISABLE
CODER_TYPE_G729ANNEXAWANNEXB	10	1†, 2, 3, or 4	Must be CODER_VAD_ENABLE ‡
CODER_TYPE_QCELP_8K *	20	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_QCELP_13K *	20	1, 2, or 3	Must be CODER_VAD_DISABLE

NOTES:
HMP = Dialogic® HMP Software; MMP = Dialogic® Multimedia Platform for AdvancedTCA
† On Dialogic® HMP Software (HMP), using fpp value of 1 is not recommended because it requires increased MIPS usage.
‡ Applications must explicitly enable VAD even though G.729a+b implicitly supports VAD
* Supported on Dialogic® Multimedia Platform for AdvancedTCA only (MMP).

IPM_AUDIO_CODER_OPTIONS_INFO

```
typedef struct ipm_audio_coder_options_info_tag
{
      unsigned int unVersion;
      unsigned int unCoderOptions;
      unsigned int unParm1;
      int nValue1;
      unsigned int unParm2;
      int nValue2;
} IPM_AUDIO_CODER_OPTIONS_INFO;
```

Description

This data structure provides additional options for audio coders, such as AMR, EVRC, and QCELP.

The EVRC and QCELP coders are supported on the Dialogic® Multimedia Platform for AdvancedTCA only.

Note: All unused fields in the IPM_AUDIO_CODER_OPTIONS_INFO structure must be set to 0.

Field Descriptions

The fields of the IPM_AUDIO_CODER_OPTIONS_INFO data structure are described as follows:

Note: For EVRC and QCELP, where the media type is MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO, all field values are ignored by the firmware.

unVersion
set to IPM_AUDIO_CODER_OPTIONS_INFO_VERSION

unCoderOptions

For AMR coders where the media type is MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO, valid values are:

- CODER_OPT_AMR_EFFICIENT - Minimize the amount of network bandwidth
  or
  CODER_OPT_AMR_OCTET - Make packet parsing easier for the AMR application

For AMR coders where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid values that can be ORed together are:

- CODER_OPT_AMR_EFFICIENT - Minimize the amount of network bandwidth
  or
  CODER_OPT_AMR_OCTET - Make packet parsing easier for the AMR application
- CODER_OPT_AMR_CMR_TRACK - Specify that the transmit bit rate should follow the CMR value in the received packet
  or
  CODER_OPT_AMR_CMR_LIMIT - Specify that the transmit bit rate should follow the CMR value in the received packet with a maximum value specified by the preferred bit rate.

For EVRC and QCELP, set to CODER_OPT_SIGNALING_OFF.

unParm1

set to a value from the eIPM_CODER_OPTION_PARMS enumeration

For AMR, set to 0.

For EVRC, where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid values are:

- CODER_OPT_INTERLEAVE_LENGTH - Reduce the listener's perception of data loss by spreading such a loss over non-consecutive vocoder frames.
- CODER_OPT_CDMA_RATE_REDUC - Rate reduction. Bit rate is varied to achieve a variety of average bit rates for more flexibility in bandwidth usage.

For QCELP, where the media type is MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO, valid values are:

- CODER_OPT_CDMA_RATE_REDUC - Rate reduction. Bit rate is varied to achieve a variety of average bit rates for more flexibility in bandwidth usage.

nValue1

The value set here is for the parameter specified in unParm1.

For AMR, set to 0.

For EVRC:

- where unParm1=CODER_OPT_INTERLEAVE_LENGTH, possible values are in the range 0 to 7. Default value is 0.
- where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0 and 4. Default value is 0.

For QCELP 8 kbps:

- where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0 and 4. Default value is 0.

For QCELP 13 kbps:

- where unParm1=CODER_OPT_CDMA_RATE_REDUC, possible values are 0, 1, 2, 3, and 4. Default value is 0.

unParm2

For AMR and EVRC, set to 0.

nValue2

For AMR and EVRC, set to 0.

Example

The following code example shows how to set options when using an EVRC coder type.

```
... /* Setup IP address here */

// Local Audio Coder

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_LOCAL_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_EVRC;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 2;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_DISABLE;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0;

unCount++;
```

```
// Remote Audio Coder

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eCoderType = CODER_TYPE_EVRC;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eFrameSize = CODER_FRAMESIZE_20;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unFramesPerPkt = 2;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.eVadEnable = CODER_VAD_DISABLE;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unCoderPayloadType = 96;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderInfo.unRedPayloadType = 0 ;

unCount++;

ipmMediaInfo.MediaData[unCount].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unVersion =
IPM_AUDIO_CODER_OPTIONS_INFO_VERSION;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unCoderOptions =
CODER_OPT_SIGNALING_OFF;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unParm1 =
CODER_OPT_INTERLEAVE_LENGTH;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.nValue1 = 6;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.unParm2 = 0;
ipmMediaInfo.MediaData[unCount].mediaInfo.AudioCoderOptionsInfo.nValue2 = 0;

unCount++;

ipmMediaInfo.unCount = unCount;
```

IPM_DIGIT_INFO

```
typedef struct ipm_digit_info_tag
{
    eIPM_DIGIT_TYPE eDigitType;             /* Type of digits */
    eIPM_DIGIT_DIRECTION eDigitDirection;   /* The direction of flow of digits */
    char          cDigits[MAX_IPM_DIGITS];  /* the digits */
    unsigned int  unNumberOfDigits;         /* Number of digits */
    unsigned int  unTimeStamp;
    unsigned int  unExpirationTime;
    unsigned int  unDuration;

} IPM_DIGIT_INFO, *PIPM_DIGIT_INFO;
```

Description

This structure is used to send and receive digits over the TDM bus using the **ipm_SendDigits( )** and **ipm_ReceiveDigits( )** functions. If your application makes a **ipm_SendDigits( )** call, it must fill in the digit type, direction, number of digits, and the actual digits to be sent. If your application makes a **ipm_ReceiveDigits( )** call, all fields are filled in upon successful return.

Field Descriptions

The fields of the IPM_DIGIT_INFO data structure are described as follows:

eDigitType
must be set to DIGIT_ALPHA_NUMERIC

The eIPM_DIGIT_TYPE data type is an enumeration which identifies the type of digit. The enumeration defines the following value:

- DIGIT_ALPHA_NUMERIC – alphanumeric digits

eDigitDirection
must be set to set to DIGIT_TDM

The eIPM_DIGIT_DIRECTION data type is an enumeration which identifies the direction of digit flow. The enumeration defines the following value:

- DIGIT_TDM – digits are sent to or received from the TDM bus

cDigits[MAX_IPM_DIGITS]
when sending digits, the actual digits to be sent; not used when receiving digits

unNumberOfDigits
number of digits being sent or received. When sending digits via **ipm_SendDigits( )**, this field indicates the number of digits to be sent; the maximum number of digits that may be sent is 16. When receiving digits via **ipm_ReceiveDigits( )**, upon return the function sets this field to the actual number of digits to be received via asynchronous events.

unTimeStamp
reserved for future use; set to 0

unExpirationTime
reserved for future use; set to 0

unDuration
reserved for future use; set to 0

IPM_EVENT_INFO

```
typedef struct ipm_event_info_tag
{
    unsigned int unCount;        /* number of following structures */
    void         *pEventData;    /* Data associated with the event */
} IPM_EVENT_INFO, *PIPM_EVENT_INFO;
```

Description

This structure is used for IP event notification. See Chapter 3, "Events" for more information.

Field Descriptions

The fields of the IPM_EVENT_INFO data structure are described as follows:

unCount
number of data structures pointed to

*pEventData
pointer to structure containing event-specific data

IPM_FAX_SIGNAL

```
typedef struct sc_tsinfo {
   eIPM_TONE eToneType;
   unsigned int unToneDuration;

} IPM_FAX_SIGNAL, *PIPM_FAX_SIGNAL;
```

Description

This structure defines the tone information detected by the gateway. IPM_FAX_SIGNAL is a child of IPM_MEDIA, which is a child of the IPM_MEDIA_INFO structure. The structure is used by the **ipm_GetLocalMediaInfo( )**, **ipm_ModifyMedia( )** and **ipm_StartMedia( )** functions.

Field Descriptions

The fields of the IPM_FAX_SIGNAL data structure are described as follows:

eToneType
: identifies type of tone to generate. The following values are defined for the eIPM_TONE enumeration:
  - TONE_NONE – no tone
  - TONE_CNG – calling (CNG) tone. Tone produced by fax machines when calling another fax machine.
  - TONE_CED – called terminal identification (CED) tone. Tone produced by fax machine when answering a call.

unToneDuration
: duration of tone to generate

IPM_INIT_FAILURE

```
typedef struct IPM_INIT_FAILURE_tag
{
      unsigned int                          unVersion;
      eIPM_RTP_PROTOCOL                     eProtocol;
      union
      {
              PIPM_NBUP_INIT_FAILURE        pNBUP;
      } data;
} IPM_INIT_FAILURE, *PIPM_INIT_FAILURE;
```

Description

The IPM_INIT_FAILURE structure describes a failure. This structure is the wrapper structure around protocol-specific structures (that is, the structure is a generic structure that contains more specific data). This structure is used by **ipm_InitSend( )** and **ipm_InitResponseSend( )**.

The INIT_IPM_INIT_FAILURE inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_INIT_FAILURE data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProtocol
: type of RTP encapsulation protocol being described. The eIPM_RTP_PROTOCOL data type is an enumeration that defines the following value:
  - RTP_PROTOCOL_NBUP – describes an Nb UP protocol

data
: data structure containing information about this initialization message. See IPM_NBUP_INIT_SEND for details.

IPM_INIT_RECEIVED

```
typedef struct IPM_INIT_RECEIVED_tag
{
      unsigned int                    unVersion;
      eIPM_RTP_PROTOCOL               eProtocol;

      union
      {
           IPM_NBUP_INIT_RECEIVED      *pNBUP;
      } data;
} IPM_INIT_RECEIVED, *PIPM_INIT_RECEIVED;
```

Description

The IPM_INIT_RECEIVED structure is the wrapper structure around protocol-specific structures (that is, the structure is a generic structure that contains more specific data). This structure is included as part of unsolicited IPMEV_INIT_RECEIVED event (enabled by EVT_INIT_RECEIVED).

The INIT_IPM_INIT_RECEIVED inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_INIT_RECEIVED data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProtocol
: type of RTP encapsulation protocol being described. The eIPM_RTP_PROTOCOL data type is an enumeration that defines the following value:
  - RTP_PROTOCOL_NBUP – describes an Nb UP protocol

data
: data structure containing information about this initialization request. See IPM_NBUP_INIT_RECEIVED structure for details.

IPM_INIT_RESPONSE

```
typedef struct IPM_INIT_RESPONSE_tag
{
      unsigned int                    unVersion;
      eIPM_RTP_PROTOCOL               eProtocol;
      union
      {
            PIPM_NBUP_INIT_RESPONSE     pNBUP;
      } data;
} IPM_INIT_RESPONSE, *PIPM_INIT_RESPONSE;
```

Description

The IPM_INIT_RESPONSE structure is the wrapper structure around protocol-specific structures (that is, the structure is a generic structure that contains more specific data). This structure is used by **ipm_InitResponseSend( )**.

The INIT_IPM_INIT_RESPONSE inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_INIT_RESPONSE data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProtocol
: type of RTP encapsulation protocol being described. The eIPM_RTP_PROTOCOL data type is an enumeration that defines the following value:
  - RTP_PROTOCOL_NBUP – describes an Nb UP protocol

data
: data structure containing information about this initialization response. See IPM_NBUP_INIT_RESPONSE structure for details.

IPM_INIT_SEND

```
typedef struct IPM_INIT_SEND_tag
{
      unsigned int                    unVersion;
      eIPM_RTP_PROTOCOL               eProtocol;
      union
      {
            PIPM_NBUP_INIT_SEND        pNBUP;
      } data;
} IPM_INIT_SEND, *PIPM_INIT_SEND;
```

Description

The IPM_INIT_SEND structure is the wrapper structure around protocol-specific structures (that is, the structure is a generic structure that contains more specific data). This structure is used by **ipm_InitSend( )**.

The INIT_IPM_INIT_SEND inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_INIT_SEND data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProtocol
: type of RTP encapsulation protocol being described. The eIPM_RTP_PROTOCOL data type is an enumeration that defines the following value:
  - RTP_PROTOCOL_NBUP – describes an Nb UP protocol

data
: data structure containing information about this initialization message. See IPM_NBUP_INIT_SEND for details.

IPM_MEDIA

```
struct IPM_MEDIA_tag
{
   eIPM_MEDIA_TYPE eMediaType;
   union
   {
      IPM_PORT_INFO           PortInfo;           /* RTP port information     */
      IPM_AUDIO_CODER_INFO    AudioCoderInfo;     /* Audio coder information  */
      IPM_FAX_SIGNAL          FaxSignal;          /* Fax signal information   */
      IPM_VIDEO_CODER_INFO    VideoCoderInfo      /* Video coder information  */
      IPM_SECURITY_INFO       SecurityInfo        /* Security information     */
      IPM_AUDIO_CODER_OPTIONS_INFO    AudioCoderOptionsInfo;
                                        /* Extended Audio Coder Information */
   }
} IPM_MEDIA, *PIPM_MEDIA;
```

Description

This structure contains information about RTP / RTCP ports, coders, security information and fax signals. It is a parent structure of:

- IPM_PORT_INFO
- IPM_AUDIO_CODER_INFO
- IPM_FAX_SIGNAL
- IPM_VIDEO_CODER_INFO
- IPM_SECURITY_INFO

This structure is a child of the IPM_MEDIA_INFO structure which is used by the **ipm_GetLocalMediaInfo( )**, **ipm_ModifyMedia( )** and **ipm_StartMedia( )** functions.

On the Dialogic® Multimedia Platform for AdvancedTCA, fields and values related to security are not supported.

Field Descriptions

The fields of the IPM_MEDIA data structure are described as follows:

eMediaType

type of media used to start or modify an IP session

The eIPM_MEDIA_TYPE data type is an enumeration which defines the following values:

Audio values:

- MEDIATYPE_AUDIO_LOCAL_CODER_INFO – local receive coder information
- MEDIATYPE_AUDIO_LOCAL_CODER_OPTIONS_INFO – options for audio coders at the local side
- MEDIATYPE_AUDIO_LOCAL_RTCP_INFO – local RTCP audio port information
- MEDIATYPE_AUDIO_LOCAL_RTP_INFO – local RTP audio port information
- MEDIATYPE_AUDIO_LOCAL_SECURITY_INFO – local receive security audio information
- MEDIATYPE_AUDIO_REMOTE_CODER_INFO – remote receive coder information
- MEDIATYPE_AUDIO_REMOTE_CODER_OPTIONS_INFO – options for audio coders at the remote side

- MEDIATYPE_AUDIO_REMOTE_RTCP_INFO – remote RTCP audio port information
- MEDIATYPE_AUDIO_REMOTE_RTP_INFO – remote RTP audio port information
- MEDIATYPE_AUDIO_REMOTE_SECURITY_INFO – remote receive audio security information

Video values:

- MEDIATYPE_VIDEO_LOCAL_CODER_INFO – local receive video coder information
- MEDIATYPE_VIDEO_LOCAL_RTCP_INFO – local RTCP video port information
- MEDIATYPE_VIDEO_LOCAL_RTP_INFO – local RTP video port information
- MEDIATYPE_VIDEO_LOCAL_SECURITY_INFO – local receive video security information
- MEDIATYPE_VIDEO_REMOTE_CODER_INFO – remote receive video coder information
- MEDIATYPE_VIDEO_REMOTE_RTCP_INFO – remote RTCP video port information
- MEDIATYPE_VIDEO_REMOTE_RTP_INFO – remote RTP video port information
- MEDIATYPE_VIDEO_REMOTE_SECURITY_INFO – remote receive video security information

Fax values:

- MEDIATYPE_FAX_SIGNAL – fax signal information to be transmitted to IP during fax transmissions
- MEDIATYPE_LOCAL_UDPTL_T38_INFO – local UDP packet T.38 information
- MEDIATYPE_REMOTE_UDPTL_T38_INFO – remote UDP packet T.38 information

Note: The following eMediaType equates are also provided for backward compatibility (the generic names are equated with audio port values). However, it is recommended that the AUDIO values be used:

```
#define MEDIATYPE_LOCAL_CODER_INFO MEDIATYPE_AUDIO_LOCAL_CODER_INFO
#define MEDIATYPE_LOCAL_RTCP_INFO MEDIATYPE_AUDIO_LOCAL_RTCP_INFO
#define MEDIATYPE_LOCAL_RTP_INFO MEDIATYPE_AUDIO_LOCAL_RTP_INFO
#define MEDIATYPE_REMOTE_CODER_INFO MEDIATYPE_AUDIO_REMOTE_CODER_INFO
#define MEDIATYPE_REPOMTE_RTCP_INFO MEDIATYPE_AUDIO_REMOTE_RTCP_INFO
#define MEDIATYPE_REPOMTE_RTP_INFO MEDIATYPE_AUDIO_REMOTE_RTP_INFO
```

PortInfo
: reference to RTP port information data structure, type IPM_PORT_INFO

AudioCoderInfo
: reference to audio coder information data structure, type IPM_AUDIO_CODER_INFO

FaxSignal
: reference to fax signal data structure, type IPM_FAX_SIGNAL

VideoCoderInfo
: reference to video coder information data structure, type IPM_VIDEO_CODER_INFO

SecurityInfo
: reference to security information data structure, type IPM_SECURITY_INFO

AudioCoderOptionsInfo
: reference to extended audio coder information data structure, type IPM_AUDIO_CODER_OPTIONS_INFO

IPM_MEDIA_INFO

```
typedef struct ipm_media_info_tag
{
    unsigned int  unCount;
    IPM_MEDIA     MediaData[MAX_MEDIA_INFO];

} IPM_MEDIA_INFO, *PIPM_MEDIA_INFO;
```

Description

This structure contains IP media session information for various kinds of media information elements, for example, RTP, RTCP and TDM. This structure is the parent of the IPM_MEDIA structure and is used by the **ipm_GetLocalMediaInfo( )**, **ipm_ModifyMedia( )**, and **ipm_StartMedia( )** functions.

Field Descriptions

The fields of the IPM_MEDIA_INFO data structure are described as follows:

unCount
number of media data structures to follow
maximum number of structures = MAX_MEDIA_INFO

MediaData
reference to IPM_MEDIA structures

IPM_OPEN_INFO

Description

This structure is used by the **ipm_Open( )** function.

Note: This structure is reserved for future use. NULL must be passed.

IPM_NBUP_INIT_FAILURE

```
typedef struct IPM_NBUP_INIT_FAILURE_tag
{
      unsigned int            unVersion;
      unsigned int            unCause;
} IPM_NBUP_INIT_FAILURE, *PIPM_NBUP_INIT_FAILURE;
```

Description

The IPM_NBUP_INIT_FAILURE structure identifies the cause of failure to establish an Nb UP session. This structure is a child of IPM_INIT_FAILURE.

The INIT_IPM_NBUP_INIT_FAILURE inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NBUP_INIT_ FAILURE data structure are described as follows:

unVersion
version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unCause
cause values describing possible reasons for failure:

- NBUP_ERR_CRCHDR – CRC error of frame header
- NBUP_ERR_CRCPYLD – CRC error of frame payload
- NBUP_ERR_UNEXP_FRAMENUM – Unexpected frame number
- NBUP_ERR_FRAMELOSS – Frame loss
- NBUP_ERR_UNKNOWN_PDUTYPE – PDU type unknown
- NBUP_ERR_UNKNOWN_PROCEDURE – Unknown procedure
- NBUP_ERR_UNKNOWN_RESERVED – Unknown reserved value
- NBUP_ERR_UNKNOWN_FIELD – Unknown field
- NBUP_ERR_FRAMESHORT – Frame too short
- NBUP_ERR_MISSING_FIELDS – Missing fields
- NBUP_ERR_UNEXP_PDUTYPE – Unexpected PDU type
- NBUP_ERR_UNEXP_PROCEDURE – Unexpected procedure
- NBUP_ERR_UNEXP_RFCI – Unexpected RFCI
- NBUP_ERR_UNEXP_VALUE – Unexpected value
- NBUP_ERR_INITIALISATION_0 – Initialization failure
- NBUP_ERR_INITIALISATION_1 – Initialization failure (network error, timer expiry)
- NBUP_ERR_INITIALISATION_2 – Initialization failure (Iu UP function error, repeated NACK).
- NBUP_ERR_RATECONTROL_FAILURE – Rate control failure
- NBUP_ERR_ERROREVENT_FAILURE – Error event failure
- NBUP_ERR_NOTSUP_TIMEALIGN – Time Alignment not supported
- NBUP_ERR_TIMEALIGN_ERROR – Requested Time Alignment not possible
- NBUP_ERR_NOTSUP_VERSION – Iu UP Mode version not supported

IPM_NACK_SENT

```
typedef struct IPM_NACK_SENT_tag
{
       unsigned int                unVersion;
       eIPM_RTP_PROTOCOL           eProtocol;
       union
       {
            PIPM_NBUP_NACK_SENT    pNBUP;
       } data;
} IPM_NACK_SENT, *PIPM_NACK_SENT;
```

Description

The IPM_NACK_SENT structure is the wrapper structure around protocol-specific structures (that is, the structure is a generic structure that contains more specific data). This structure is included as part of unsolicited IPMEV_NACK_SENT event (enabled by EVT_NACK_SENT).

The INIT_IPM_NACK_SENT inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NACK_SENT data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProtocol
: type of RTP encapsulation protocol being described. The eIPM_RTP_PROTOCOL data type is an enumeration that defines the following value:
  - RTP_PROTOCOL_NBUP – describes an Nb UP protocol

data
: data structure containing information about why the NACK response was sent. See IPM_NBUP_NACK_SENT structure for details.

IPM_NBUP_INIT_RECEIVED

Description

The IPM_NBUP_INIT_RECEIVED structure and related macros are aliases of the IPM_NBUP_INIT_SEND structure and macros. The data in the incoming Nb UP request will always be the same as the data in the outbound Nb UP request. They are defined as follows:

```
#define IPM_NBUP_INIT_RECEIVED                  IPM_NBUP_INIT_SEND
#define INIT_IPM_NBUP_INIT_RECEIVED             INIT_IPM_NBUP_INIT_SEND
#define FREE_IPM_NBUP_INIT_RECEIVED             FREE_IPM_NBUP_INIT_SEND
```

IPM_NBUP_INIT_RESPONSE

```
typedef struct IPM_NBUP_INIT_RESPONSE_tag
{
       unsigned int                    unVersion;
       eIPM_NBUP_INIT_RESPONSE         eResponse;         /* Type of response */
} IPM_NBUP_INIT_RESPONSE, *PIPM_NBUP_INIT_RESPONSE;
```

Description

The IPM_NBUP_INIT_RESPONSE structure identifies the type of response to send to an Nb UP request. This structure is a child of the IPM_INIT_RESPONSE structure.

The INIT_IPM_NBUP_INIT_RESPONSE inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NBUP_INIT_RESPONSE data structure is described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eResponse
: response to an Nb UP request. The eIPM_NBUP_INIT_RESPONSE_TYPE data type is an enumeration that defines the following values:
  - NBUP_INIT_RESPONSE_ACK – acknowledge the response and accept the invitation
  - NBUP_INIT_RESPONSE_NACK – send a NACK response to deny the invitation

IPM_NBUP_INIT_SEND

```
typedef struct ipm_NBUP_INIT_SEND_tag
{
    unsigned int                unVersion;
    unsigned int                unSubFlowCount;     /* Number of subflows per RFCI */
    unsigned int                unRFCICount;        /* Number of RFCI structures */
    IPM_NBUP_RFCI_INFO          *pRFCIs;            /* RFCI structures */
} IPM_NBUP_INIT_SEND, *PIPM_NBUP_INIT_SEND;
```

Description

The IPM_NBUP_INIT_SEND structure describes Nb UP initialization items. Since each RFCI must have the same number of subflows, the count is maintained here. This structure is a child of IPM_INIT_SEND structure. The IPM_NBUP_INIT_RECEIVED structure is an alias of the IPM_NBUP_INIT_SEND.

The INIT_IPM_NBUP_INIT_SEND inline function is provided to initialize any dynamic memory needed by the structure and to fill in the values provided by the user.

The FREE_IPM_NBUP_INIT_SEND inline function is provided to de-allocate any dynamic memory allocated by INIT.

Field Descriptions

The fields of the IPM_NBUP_INIT_SEND data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

unSubFlowCount
: number of SubFlow structures contained in each RFCI structure

unRFCICount
: number of RFCI structures contained in pRFCIs

pRFCIs
: pointer to an array of IPM_NBUP_RFCI_INFO structures

IPM_NBUP_NACK_SENT

```
typedef struct IPM_NBUP_NACK_SENT_tag
{
       unsigned int                    unVersion;
       eIPM_NBUP_NACK_PROCEDURE        eProcedure;
       unsigned int                    unCause;
} IPM_NBUP_NACK_SENT, *PIPM_NBUP_NACK_SENT;
```

Description

The IPM_NBUP_NACK_SENT structure identifies the type of NACK response to send to an Nb UP request. This structure is a child of the IPM_NACK_SENT structure.

The INIT_IPM_NBUP_NACK_SENT inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NBUP_NACK_SENT data structure are described as follows:

unVersion
version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProcedure
enumeration describing possible reasons for the NACK response. The eIPM_NBUP_NACK_PROCEDURE data type is an enumeration that defines the following values:

- NBUP_NACK_INITIALIZE – error during initialization
- NBUP_NACK_RATE_CONTROL – rate control problem
- NBUP_NACK_TIME_ALIGNMENT – time alignment error

unCause
cause values describing possible reasons for NACK cause.

- NBUP_ERR_CRCHDR – CRC error of frame header
- NBUP_ERR_CRCPYLD – CRC error of frame payload
- NBUP_ERR_UNEXP_FRAMENUM – Unexpected frame number
- NBUP_ERR_FRAMELOSS – Frame loss
- NBUP_ERR_UNKNOWN_PDUTYPE – PDU type unknown
- NBUP_ERR_UNKNOWN_PROCEDURE – Unknown procedure
- NBUP_ERR_UNKNOWN_RESERVED – Unknown reserved value
- NBUP_ERR_UNKNOWN_FIELD – Unknown field
- NBUP_ERR_FRAMESHORT – Frame too short
- NBUP_ERR_MISSING_FIELDS – Missing fields
- NBUP_ERR_UNEXP_PDUTYPE – Unexpected PDU type
- NBUP_ERR_UNEXP_PROCEDURE – Unexpected procedure
- NBUP_ERR_UNEXP_RFCI – Unexpected RFCI
- NBUP_ERR_UNEXP_VALUE – Unexpected value
- NBUP_ERR_INITIALISATION_0 – Initialization failure
- NBUP_ERR_INITIALISATION_1 – Initialization failure (network error, timer expiry)

- NBUP_ERR_INITIALISATION_2 – Initialization failure (Iu UP function error, repeated NACK).
- NBUP_ERR_RATECONTROL_FAILURE – Rate control failure
- NBUP_ERR_ERROREVENT_FAILURE – Error event failure
- NBUP_ERR_NOTSUP_TIMEALIGN – Time Alignment not supported
- NBUP_ERR_TIMEALIGN_ERROR – Requested Time Alignment not possible
- NBUP_ERR_NOTSUP_VERSION – Iu UP Mode version not supported

IPM_NBUP_PROFILE_INFO

```
typedef struct ipm_nbup_profile_info_tag
{
      unsigned int                    unVersion;
      eIPM_NBUP_PROFILE_TYPE          eProfileType;
} IPM_NBUP_PROFILE_INFO, *PIPM_NBUP_PROFILE_INFO;
```

Description

The IPM_NBUP_PROFILE_INFO structure describes the type of Nb UP profile being requested in the call to **ipm_StartMedia( )**. This structure is a child of IPM_MEDIA.

The INIT_IPM_NBUP_PROFILE_INFO inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NBUP_PROFILE_INFO data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eProfileType
: the type of profile to set up in **ipm_StartMedia( )**. The eIPM_NBUP_PROFILE_TYPE data type is an enumeration that defines the following value:
  - NBUP_PROFILE_3G324M – a 3G-324M type connection

IPM_NBUP_RFCI_INFO

```
typedef struct ipm_NBUP_RFCI_INFO_tag
{
       unsigned int               unVersion;
       unsigned char              ucID;
       IPM_NBUP_SUBFLOW_INFO      *pSubFlows;
} IPM_NBUP_RFCI_INFO, *PIPM_NBUP_RFCI_INFO;
```

Description

This structure describes the RFCI items in the Nb UP initialization message. This structure is a child of IPM_NBUP_INIT_SEND structure.

The INIT_IPM_NBUP_RFCI_INFO inline function is provided to initialize any dynamic memory needed by the structure and to fill in the values provided by the user.

The FREE_IPM_NBUP_RFCI_INFO inline function is provided to de-allocate any dynamic memory allocated by INIT.

Field Descriptions

The fields of the IPM_NBUP_RFCI_INFO data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

ucID
: identifier of this subflow

pSubFlows
: a list of subflow structures. The count of subflows is maintained by the value in the IPM_NBUP_SUBFLOW_INFO structure.

IPM_NBUP_SUBFLOW_INFO

```
typedef struct IPM_NBUP_SUBFLOW_INFO_tag
{
      unsigned int                    unVersion;
      eIPM_NBUP_FLOW_SIZE             eFlowSize;
} IPM_NBUP_SUBFLOW_INFO, *PIPM_NBUP_SUBFLOW_INFO;
```

Description

This structure describes the attributes of a subflow. This structure is a child of IPM_NBUP_RFCI_INFO structure.

The INIT_IPM_NBUP_SUBFLOW_INFO inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_NBUP_SUBFLOW_INFO data structure are described as follows:

unVersion
: version of the data structure. Used to ensure that an application is binary compatible with future changes to this data structure. Use the inline function to initialize this field to the current version.

eFlowSize
: number of bits to use for each of the flows in the IPM_NBUP_SUBFLOW_INFO data type structure. The eIPM_NBUP_FLOW_SIZE data type is an enumeration that defines the following values:
  - NBUP_FLOW_SIZE_0_BITS – use 0 bits
  - NBUP_FLOW_SIZE_320_BITS – use 320 bits

SC_TSINFO

```
typedef struct sc_tsinfo {
   unsigned long   sc_numts;
   long            *sc_tsarrayp;
} SC_TSINFO;
```

Description

This structure defines the TDM bus (CT Bus) time slot information. It is used by **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_StartMedia( )**, and **ipm_GetLocalMediaInfo( )**.

Field Descriptions

The fields of the SC_TSINFO data structure are described as follows:

sc_numts
: number of time slots to follow; must be set to 1 for this release

sc_tsarrayp
: time slot ID number

IPM_PARM_INFO

```
typedef struct ipm_param_info_tag
{
    eIPM_PARM     eParm;          /* the parameter to set or get */
    void          *pvParmValue;   /* pointer to value of parameter */
} IPM_PARM_INFO, *PIPM_PARM_INFO;
```

Description

This structure is used to set or retrieve parameters for an IP channel. The structure is used by the **ipm_GetParm( )** and **ipm_SetParm( )** functions.

Field Descriptions

The fields of the IPM_PARM_INFO data structure are described as follows:

eIPM_PARM
type of parameter to set or get. See Table 3 for supported types and corresponding values.

pvParmValue
pointer to the value of the parameter. See Table 3 for supported values for each parameter type.

Table 3. eIPM_PARM Parameters and Values

eIP_PARM Define	Description and Values
PARMCH_DTMFXFERMODE	Specifies DTMF transfer mode Type: eIPM_DTMFXFERMODE (enumeration). Values: • DTMFXFERMODE_INBAND – in-band (default) • DTMFXFERMODE_OUTOFBAND – out-of-band • DTMFXFERMODE_RFC2833 – RFC2833 **Note:** In order for DTMF event reporting to occur, you must set out-of-band signaling on the receive side.
PARMCH_ECACTIVE	Enables/disables echo cancellation. Type: eIPM_ACTIVE (enumeration). Values: • ECACTIVE_OFF (default) • ECACTIVE_ON
PARMCH_ECHOTAIL	Specifies echo tail length value. Type: eIPM_ECHO_TAIL (enumeration) Supported values: • ECHO_TAIL_NONE • ECHO_TAIL_8 • ECHO_TAIL_16 (default) • ECHO_TAIL_32 • ECHO_TAIL_64

Table 3. eIPM_PARM Parameters and Values (Continued)

eIP_PARM Define	Description and Values
PARMCH_NOTIFY_EXPIRE_KEY_AUDIO	Supported on Dialogic® HMP Software only. For Secure RTP (SRTP), specifies the time before encryption key expiry at which notification is provided that the encryption key for the audio media type is about to expire. Type: Integer. Values: In 100 ms units. Default: 1 (100 ms). **Note:** If the default setting for a given device is changed at any time after download, the new settings for that device are persistent beyond the lifetime of the application (even if the device is closed and re-opened). The settings stay in effect until either the next download or until the settings are overridden with another ipm_SetParm( ) call.
PARMCH_NOTIFY_EXPIRE_KEY_VIDEO	Supported on Dialogic® HMP Software only. For Secure RTP (SRTP), specifies the time before encryption key expiry at which notification is provided that the encryption key for the video media type is about to expire. Type: Integer. Values: In 100 ms units. Default: 1 (100 ms). **Note:** If the default setting for a given device is changed at any time after download, the new settings for that device are persistent beyond the lifetime of the application (even if the device is closed and re-opened). The settings stay in effect until either the next download or until the settings are overridden with another ipm_SetParm( ) call.
PARMCH_ECNLP_ACTIVE	Supported on Dialogic® HMP Software only. Enables/disables Non-Linear Processing (NLP) value for echo cancellation. NLP is a process in which signals that have a level below a defined threshold are replaced by comfort noise and signals that have a level above the threshold are passed unmodified. Type: Integer. Values: 0 or 1. Default: 1.
PARMCH_RFC2833EVT_RX_PLT	Specifies RFC2833 event receive payload type. Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMCH_RFC2833EVT_TX_PLT	Specifies RFC2833 event transmit payload type. Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMBD_RTCPAUDIO_INTERVAL	Specifies the RTCP calculation frequency for audio streams. Type: integer. Valid Values: 1 to 15. Default: 5. **Note:** 1. This parameter is a board-level parameter; the setting is applied to all IP devices on a virtual board. 2. If the value is changed, the new value is immediately applied to the virtual board. 3. This parameter applies to audio RTP and RTCP packets only.

Table 3. eIPM_PARM Parameters and Values (Continued)

eIP_PARM Define	Description and Values
PARMBD_RTPAUDIO_PORT_BASE	Specifies the RTP base port number for audio streams. Type: integer. Valid Values: 0 to 65535. Default: 49152. **Note:** 1. This parameter is a board-level parameter; the setting is applied to all IP devices on a virtual board. 2. If the RTP base port number is changed while a virtual board's channels are in an active call (streaming), the new value does not take effect on the active channels until the calls end. However, modifying the RTP base port number on an IP device with active channels causes unexpected behavior on the active calls. 3. This parameter applies to audio RTP and RTCP packets only.
PARMCH_RX_ADJVOLUME	Specifies volume level adjustment for inbound (from IP) side in 1 dB increments. Type: integer. Valid values: -32 to 31. Default: 0 (no adjustment).
PARMCH_TX_ADJVOLUME	Specifies volume level adjustment for outbound (to IP) side in 1 dB increments. Type: integer. Valid values: -32 to 31. Default: 0 (no adjustment).
PARMCH_TOS	Specifies type of service in IPv4 headers. This can be either a 7-bit TOS field or a 6-bit DSCP field for Differentiated Services per RFC2474. Type: char. Valid values: 0 to 255. Default: 0.
PARMCH_TTL	Specifies time-to-live for multicast. Type: char. Valid values: 0 to 255. Default: 1.

IPM_PORT_INFO

```
typedef struct ipm_port_info_tag
{
   unsigned int   unPortId;                  /* The Port ID */
   char           cIPAddress[IP_ADDR_SIZE];   /* IP Address  */

} IPM_PORT_INFO, *PIPM_PORT_INFO;
```

Description

This structure contains RTP, RTCP, and T.38 UDP port properties. It is a child of IPM_MEDIA, which is a child of the IPM_MEDIA_INFO structure that is used by **ipm_GetLocalMediaInfo( )** **ipm_ModifyMedia( )** and **ipm_StartMedia( )**.

Field Descriptions

The fields of the IPM_PORT_INFO data structure are described as follows:

unPortId
port identifier

cIPAddress[IP_ADDR_SIZE]
null-terminated IP address of the port in standard dotted decimal string format; for example, 192.168.0.1

Note: Avoid setting IP address 0.0.0.0 when using **ipm_StartMedia( )** because this may cause a hung port.

IPM_QOS_ALARM_DATA

```
typedef struct ipm_qos_alarm_data_tag
{
   eIPM_QOS_TYPE     eQoSType;          /* The QOS parameter type  */
   eIPM_ALARM_STATE  eAlarmState;       /* indicate if On/Off */

} IPM_QOS_ALARM_DATA, *PIPM_QOS_ALARM_DATA;
```

Description

This structure is used to retrieve data associated with QoS alarms, as reported in IPMEV_QOS_ALARM events. It is also a child of the IPM_QOS_ALARM_STATUS structure, which is used by the **ipm_GetQoSAlarmStatus( )** and **ipm_ResetQoSAlarmStatus( )** functions.

The library generates a IPMEV_QOS_ALARM alarm event with ALARM_STATE_ON when a QoS fault threshold is exceeded, and it generates a generates a QoS alarm event with ALARM_STATE_OFF when the fault measurement returns to a sub-threshold level.

Field Descriptions

The fields of the IPM_QOS_ALARM_DATA data structure are described as follows:

eQoSType
identifies the alarm event that has occurred

The eIPM_QOS_TYPE data type is an enumeration which defines the following values:

- QOSTYPE_JITTER – QoS alarm for excessive average jitter
- QOSTYPE_LOSTPACKETS – excessive lost packets
- QOSTYPE_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE_RTPTIMEOUT – RTP inactivity

The following values are defined for Secure RTP (SRTP):

- QOSTYPE_SEC_AUTH_FAIL_AUDIO – excessive number of audio packets failing authentication in a given time interval
- QOSTYPE_SEC_AUTH_FAIL_VIDEO – excessive number of video packets failing authentication in a given time interval
- QOSTYPE_SEC_PKT_REPLAY_AUDIO – excessive number of audio replay packets detected in a given time interval
- QOSTYPE_SEC_PKT_REPLAY_VIDEO – excessive number of video replay packets detected in the given time interval
- QOSTYPE_SEC_MKI_NOMATCH_AUDIO – excessive number of audio packets with an MKI that did not match the list of MKI values on the receiving side in a given time interval
- QOSTYPE_SEC_MKI_NOMATCH_VIDEO – excessive number of video packets with an MKI that did not match the list of MKI values on the receiving side in a given time interval

eAlarmState
alarm on / off flag

The eIPM_ALARM_STATE data type is an enumeration which defines the following values:

- ALARM_STATE_OFF – alarm is OFF
- ALARM_STATE_ON – alarm is ON

IPM_QOS_ALARM_STATUS

```
typedef struct ipm_qos_alarm_status_tag
{
   unsigned int unAlarmCount;
   IPM_QOS_ALARM_DATA QoSData[MAX_ALARM];

} IPM_QOS_ALARM_STATUS, *PIPM_QOS_ALARM_STATUS;
```

Description

This structure contains the status of QoS alarms for an IP channel. It is the parent of IPM_QOS_ALARM_DATA and is used by **ipm_GetQoSAlarmStatus( )** and **ipm_ResetQoSAlarmStatus( )**.

Field Descriptions

The fields of the IPM_QOS_ALARM_STATUS data structure are described as follows:

unAlarmCount
: number of QoSData structures to follow
: maximum number of alarms = MAX_ALARM

QoSData
: reference to alarm data information structure IPM_QOS_ALARM_DATA

IPM_QOS_SESSION_INFO

```
typedef struct ipm_qos_session_info_tag
{
      eIPM_QOS_TYPE  eQoSType;
      unsigned int   unData;

} IPM_QOS_SESSION_INFO, *PIPM_QOS_SESSION_INFO;
```

Description

This structure reports statistical Quality of Service information for an IP session. It is a child of the IPM_SESSION_INFO structure which is filled in when **ipm_GetSessionInfo( )** returns successfully.

Field Descriptions

The fields of the IPM_QOS_SESSION_INFO data structure are described as follows:

eQoSType
identifies the QoS alarm to retrieve statistics for

The eIPM_QOS_TYPE data type is an enumeration which defines the following values:

- QOSTYPE_JITTER – average jitter (in msec) since beginning of call
- QOSTYPE_LOSTPACKETS – percentage of lost packets since beginning of call
- QOSTYPE_RTCPTIMEOUT – RTCP inactivity (in msec)
- QOSTYPE_RTPTIMEOUT – RTP inactivity (in msec)

The following values are defined for Secure RTP (SRTP):

- QOSTYPE_SEC_AUTH_FAIL_AUDIO – number of audio packets failing authentication since the beginning of the call
- QOSTYPE_SEC_AUTH_FAIL_VIDEO – number of video packets failing authentication since the beginning of the call
- QOSTYPE_SEC_PKT_REPLAY_AUDIO – number of audio replay packets detected since the beginning of the call
- QOSTYPE_SEC_PKT_REPLAY_VIDEO – number of video replay packets detected since the beginning of the call
- QOSTYPE_SEC_MKI_NOMATCH_AUDIO – number of audio packets with an MKI that did not match the list of MKI values on the receiving side since the beginning of the call
- QOSTYPE_SEC_MKI_NOMATCH_VIDEO – number of video packets with an MKI that did not match the list of MKI values on the receiving side since the beginning of the call

unData
value of the QoS parameter

IPM_QOS_THRESHOLD_DATA

```
typedef struct ipm_qos_threshold_data_tag
{
    eIPM_QOS_TYPE eQoSType;
    unsigned int  unTimeInterval;
    unsigned int  unDebounceOn;
    unsigned int  unDebounceOff;
    unsigned int  unFaultThreshold;
    unsigned int  unPercentSuccessThreshold;
    unsigned int  unPercentFailThreshold;

} IPM_QOS_THRESHOLD_DATA, *PIPM_QOS_THRESHOLD_DATA;
```

Description

This structure contains the threshold values for QoS alarms for an IP channel. It is a child of the IPM_QOS_THRESHOLD_INFO structure which is used by **ipm_GetQoSThreshold( )** and **ipm_SetQoSThreshold( )**. When enabling a QoS alarm, default threshold and timing values as shown in Table 4 will be used unless **ipm_SetQoSThreshold( )** is used to set non-default values. Note that when an application sets a specific value for any field of a IPM_QOS_THRESOLD structure, it must explicitly set *all* fields in the structure even when default values are desired for some of the fields.

Field Descriptions

The fields of the IPM_QOS_THRESHOLD_DATA data structure are described as follows:

eQoSType
: type of QoS parameter to measure

  The eIPM_QOS_TYPE data type is an enumeration which defines the following values:

- QOSTYPE_JITTER – jitter
- QOSTYPE_LOSTPACKETS – lost packets
- QOSTYPE_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE_RTPTIMEOUT – RTP inactivity
- QOSTYPE_SEC_AUTH_FAIL_AUDIO – authentication failure on audio packets
- QOSTYPE_SEC_AUTH_FAIL_VIDEO – authentication failure on video packets
- QOSTYPE_SEC_PKT_REPLAY_AUDIO – detection of audio packet replay
- QOSTYPE_SEC_PKT_REPLAY_VIDEO – detection of video packet replay
- QOSTYPE_SEC_MKI_NOMATCH_AUDIO – unknown MKI value in audio packets
- QOSTYPE_SEC_MKI_NOMATCH_VIDEO – unknown MKI value in video packets

unTimeInterval
: time interval (in ms) between successive parameter measurements. Value should be set to a multiple of 100; other values are rounded to the nearest hundred.

  Note: Value must be greater than unFaultThreshold for the jitter QoS type.

unDebounceOn
: time interval for detecting potential alarm fault condition. Must be set to a value that is a multiple of unTimeInterval; other values are rounded down to the next lower multiple of unTimeInterval.

  Note: This field is not used for RTCP and RTP Timeout alarms and must be set to 0.

unDebounceOff

time interval for detecting potential alarm non-fault condition. Must be set to a value that is a multiple of unTimeInterval; other values are rounded down to the next lower multiple of unTimeInterval.

Note: This field is not used for RTCP and RTP Timeout alarms and must be set to 0.

unFaultThreshold

fault threshold parameter. The meaning and value range of this field depends on the QoS Type:

- QOSTYPE_JITTER – allowable average jitter, in ms. Range: 0 to 1000 (ms)
- QOSTYPE_LOSTPACKET – allowable percentage of lost packets. Range: 0 to 100 (%)
- QOSTYPE_RTCPTIMEOUT – allowable RTCP inactive interval before an alarm is sent, in units of 100 ms. Range: 50 to 1200 (x100 ms)
- QOSTYPE_RTPTIMEOUT – allowable RTP inactive interval before an alarm is sent, in units of 100 ms. Range: 50 to 1200 (x100 ms)
- QOSTYPE_SEC_AUTH_FAIL_AUDIO – allowed number of audio packet authentication failures before an alarm is sent.
- QOSTYPE_SEC_AUTH_FAIL_VIDEO – allowed number of video packet authentication failures before an alarm is sent.
- QOSTYPE_SEC_PKT_REPLAY_AUDIO – allowed number of audio replay packets detected before an alarm is sent.
- QOSTYPE_SEC_PKT_REPLAY_VIDEO – allowed number of video replay packets detected before an alarm is sent.
- QOSTYPE_SEC_MKI_NOMATCH_AUDIO – allowed number of audio packets with MKI mismatches detected before an alarm is sent.
- QOSTYPE_SEC_MKI_NOMATCH_VIDEO – allowed number of video packets with MKI mismatches detected before an alarm is sent.

unPercentSuccessThreshold

percentage of poll instances in unDebounceOff time interval that the fault threshold must not be exceeded before an “alarm off” event is sent. Allowed values correspond to multiples of the ratio of unDebounceOff to unTimeInterval (i.e., the inverse of the number of poll instances) expressed as an integer percentage; other values are truncated to the next lower percentage multiple.

Note: This parameter is not used for RTCP and RTP Timeout alarms and must be set to 0.

unPercentFailThreshold

percentage of poll instances in unDebounceOn time interval that the fault threshold must be exceeded before an “alarm on” event is sent. Allowed values correspond to multiples of the ratio of unDebounceOn to unTimeInterval (i.e., the inverse of the number of poll instances) expressed as a integer percentage; other values are truncated to the next lower percentage multiple.

Note: This parameter is not used for RTCP and RTP Timeout alarms and must be set to 0.

Table 4. Quality of Service Parameter Defaults for Dialogic® Host Media Processing (HMP) Software

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold [1]	% Success Threshold	% Fail Threshold
Jitter	5000	20000	60000	60 (ms)	25	25
Lost Packets	1000	10000	10000	20 (%)	40	40
RTCP Timeout	1000	0	0	250 (x100ms = 25sec)	0	0
RTP Timeout	1000	0	0	1200 (x100ms = 120sec)	0	0
Audio Authentication Failure	1000	10000	10000	10 packets	20	20
Video Authentication Failure	1000	10000	10000	10 packets	20	20
Audio Packet Replay	1000	10000	10000	10 packets	20	20
Video Packet Replay	1000	10000	10000	10 packets	20	20
Audio MKI No Match	1000	10000	10000	10 packets	20	20
Video MKI No Match	1000	10000	10000	10 packets	20	20

Notes:
1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.

QoS debouncing is calculated as an integer number of parameter measurements that must exceed (or fall below) the fault threshold within the debounce interval before an alarm-on (or alarm-off) event is generated. The calculation uses the following formulas:

For QoS alarm-on debouncing:

count = int(int(unDebounceOn/unTimeInterval) * (unPercentFailThreshold/100))

For QoS alarm-off debouncing:

count = int(int(unDebounceOff/unTimeInterval) * (unPercentSuccessThreshold/100))

IPM_QOS_THRESHOLD_INFO

```
typedef struct ipm_qos_threshold_info_tag
{
   unsigned int unCount;
   IPM_QOS_THRESHOLD_DATA QoSThresholdData[MAX_QOS_THRESHOLD];

} IPM_QOS_THRESHOLD_INFO, *PIPM_QOS_THRESHOLD_INFO;
```

Description

This structure is used to set and get the threshold values for QoS alarms for a single IP channel. It is the parent of IPM_QOS_THRESHOLD_DATA and is used by **ipm_GetQoSThreshold( )** and **ipm_SetQoSThreshold( )**.

Field Descriptions

The fields of the IPM_QOS_THRESHOLD_INFO data structure are described as follows:

unCount
: number of IPM_QOS_THRESHOLD_DATA structures to follow;
maximum = MAX_QOS_THRESHOLD

QosThresholdData
: array of structures containing alarm trigger settings

IPM_RFC2833_SIGNALID_INFO

```
typedef struct ipm_rfc2833_signalid_info_tag
{
   eIPM_RFC2833_SIGNAL_ID  eSignalID;
   eIPM_SIGNAL_STATE       eState;

} IPM_RFC2833_SIGNALID_INFO;
```

Description

This structure is used to identify RFC 2833-compliant signals that are received from the IP network. It is used in the IPMEV_RFC2833SIGNALRECEIVED events that are generated when RFC2833 signals are detected on IP.

Field Descriptions

The fields of the IPM_RFC2833_SIGNALID_INFO data structure are described as follows:

eSignalID
Identifies the RFC2833 signal. The eIPM_RFC2833_SIGNAL_ID is an enumeration with values as listed in Table 5.

Table 5. eIPM_RFC2833_SIGNAL_ID Values

Name	Value (Hex)
SIGNAL_ID_EVENT_DTMF_0	0x0
SIGNAL_ID_EVENT_DTMF_1	0x1
SIGNAL_ID_EVENT_DTMF_2	0x2
SIGNAL_ID_EVENT_DTMF_3	0x3
SIGNAL_ID_EVENT_DTMF_4	0x4
SIGNAL_ID_EVENT_DTMF_5	0x5
SIGNAL_ID_EVENT_DTMF_6	0x6
SIGNAL_ID_EVENT_DTMF_7	0x7
SIGNAL_ID_EVENT_DTMF_8	0x8
SIGNAL_ID_EVENT_DTMF_9	0x9
SIGNAL_ID_EVENT_DTMF_STAR	0xa
SIGNAL_ID_EVENT_DTMF_POUND	0xb
SIGNAL_ID_EVENT_DTMF_A	0xc
SIGNAL_ID_EVENT_DTMF_B	0xd
SIGNAL_ID_EVENT_DTMF_C	0xe
SIGNAL_ID_EVENT_DTMF_D	0xf

eState
This field is not used when receiving RFC2833 signals.

IPM_RTCP_SESSION_INFO

```
typedef struct ipm_rtcp_session_info_tag
{
   unsigned int    unLocalSR_TimeStamp;
   unsigned int    unLocalSR_TxPackets;
   unsigned int    unLocalSR_TxOctets;
   unsigned int    unLocalSR_SendIndication;
   unsigned int    unLocalRR_FractionLost;
   unsigned int    unLocalRR_CumulativeLost;
   unsigned int    unLocalRR_SeqNumber;
   unsigned int    unLocalRR_ValidInfo;
   unsigned int    unRemoteSR_TimeStamp;
   unsigned int    unRemoteSR_TxPackets;
   unsigned int    unRemoteSR_TxOctets;
   unsigned int    unRemoteSR_SendIndication;
   unsigned int    unRemoteRR_FractionLost;
   unsigned int    unRemoteRR_CumulativeLost;
   unsigned int    unRemoteRR_SeqNumber;
   unsigned int    unRemoteRR_ValidInfo;

} IPM_RTCP_SESSION_INFO, *PIPM_RTCP_SESSION_INFO;
```

Description

This structure contains RTCP information for the session. It is a child of the IPM_SESSION_INFO structure which is filled in when **ipm_GetSessionInfo( )** returns successfully.

Field Descriptions

The fields of the IPM_RTCP_SESSION_INFO data structure are described as follows:

unLocalSR_TimeStamp
: time stamp of the RTCP packet transmission from the local sender

unLocalSR_TxPackets
: number of packets sent by the local sender

unLocalSR_TxOctets
: number of bytes sent by the local sender

unLocalSR_SendIndication
: local sender report has changed since the last transmission. Values may be either:
  - FALSE
  - TRUE

unLocalRR_FractionLost
: percentage of packets lost, as computed by the local receiver

unLocalRR_CumulativeLost
: number of packets lost, as computed by the local receiver

unLocalRR_SeqNumber
: last sequence number received from the local receiver

unLocalRR_ValidInfo
: reserved for future use

unRemoteSR_TimeStamp
time stamp of the RTCP packet transmission from the remote sender

unRemoteSR_TxPackets
number of packets sent by the remote sender

unRemoteSR_TxOctets
number of bytes sent by the remote sender

unRemoteSR_SendIndication
remote sender report has changed since the last transmission. Values may be either:

- FALSE
- TRUE

unRemoteRR_FractionLost
percentage of packets lost, as computed by the remote receiver

unRemoteRR_CumulativeLost
number of packets lost, as computed by the remote receiver

unRemoteRR_SeqNumber
last sequence number received from the remote receiver

unRemoteRR_ValidInfo
reserved for future use

IPM_SECURITY_BASE64_KEY

```
typedef struct ipm_security_base64_key_tag
{
    unsigned int    unVersion;
    char            *pcMasterBase64Key;
} IPM_SECURITY_BASE64_KEY, *PIPM_SECURITY_BASE64_KEY;
```

Description

This structure contains information relating to Base64-encoded security keys. When the security key type being used is Base64, this structure is a child of the IPM_SECURITY_KEY structure, which in turn is a child of the IPM_SECURITY_INFO structure, which in turn is a child of the IPM_MEDIA structure, which in turn is a child of the IPM_MEDIA_INFO structure used by the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions.

The INIT_IPM_SECURITY_BASE64_KEY inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_SECURITY_BASE64_KEY data structure are described as follows:

unVersion
the version of the data structure

pcMasterBase64Key
a character string that is the Base64-encoded master key

Note: pcMasterBase64Key should point to a character array whoes length is at least that defined by (unMasterKeyLength + unMasterSaltKeyLength)/6 and rounded up.

IPM_SECURITY_BINARY_KEY

```
typedef struct ipm_security_binary_key_tag

{
    unsigned int    unVersion;
    char            *pcMasterKey;
    char            *pcMasterSaltKey;
} IPM_SECURITY_BINARY_KEY, *PIPM_SECURITY_BINARY_KEY;
```

Description

This structure contains information relating to binary security keys. When the security key type being used is binary, this structure is a child of the IPM_SECURITY_KEY structure, which in turn is a child of the IPM_SECURITY_INFO structure, which in turn is a child of the IPM_MEDIA structure, which in turn is a child of the IPM_MEDIA_INFO structure used by the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions.

The IPM_SECURITY_BINARY_KEY inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_SECURITY_BINARY_KEY data structure are described as follows:

unVersion
the version of the data structure

pcMasterKey
a character string that is the binary master key

Note: pcMasterKey should point to a character array whoes length is at least that defined by unMasterKeyLength/8.

pcMasterSaltKey
a character string that is the binary master salt key

Note: pcMasterSaltKey should point to a character array whose length is at least that defined by unMasterSaltKeyLength/8.

IPM_SECURITY_INFO

```
typedef struct ipm_security_info_tag {
    unsigned int            unVersion;
    unsigned int            unNumKeys;
    eIPM_SECURITY_INFO_MODE eInfoMode;
    IPM_SECURITY_KEY        *pKeys;
    IPM_SRTP_PARMS          *pParms;
} IPM_SECURITY_INFO, *PIPM_SECURITY_INFO;
```

Description

This structure contains security information. It is a child of the IPM_MEDIA structure, which in turn is a child of the IPM_MEDIA_INFO structure used by the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions.

The INIT_IPM_SECURITY_INFO inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_SECURITY_INFO data structure are described as follows:

unVersion
: version of the IPM_SECURITY_INFO structure

unNumKeys
: number of keys passed

eInfoMode
: specifies the mode of operation. Currently, there are two modes of operation as defined in the eIPM_SECURITY_INFO_MODE enumeration:

```
typedef enum eipm_security_info_mode_tag
{
    IPM_SECURITY_INFO_MODE_IMMEDIATE          = 1,
    IPM_SECURITY_INFO_MODE_WAITFOREXPIRATION   = 2
} eIPM_SECURITY_INFO_MODE;
```

The meaning of each value is as follows:

- IPM_SECURITY_INFO_MODE_IMMEDIATE (default) – apply new security keys immediately
- IPM_SECURITY_INFO_MODE_WAITFOREXPIRATION – wait until all the specified key lifetimes have expired before applying new keys

pKeys
: pointer to array of IPM_SECURITY_KEY structures. The number of keys this pointer can point to is specified in unNumKeys.

pParms
: pointer to an IPM_SRTP_PARMS structure. The same secure RTP (SRTP) parameter values are applied to all the keys specified.

IPM_SECURITY_KEY

```
typedef struct ipm_security_key_tag
{
    UINT64                  unSRTPLifeTime;
    unsigned int            unVersion;
    unsigned int            unSRTCPLifeTime;
    unsigned int            unMkiLength;
    unsigned int            unMkiValue;
    unsigned int            unMasterKeyLength;
    unsigned int            unMasterSaltKeyLength;
    eIPM_SECURITY_KEY_TYPE  eKeyType;
    void                    *pvMasterKey;
} IPM_SECURITY_KEY, *PIPM_SECURITY_KEY;
```

Description

This structure contains security key information. It is used by the **ipm_SecurityGenMasterKeys( )** function when generating master and salt keys. It is also a child of the IPM_SECURITY_INFO structure, which in turn is a child of the IPM_MEDIA structure, which in turn is a child of the IPM_MEDIA_INFO structure used by the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions.

Currently, two types of security keys are supported:

- Binary Keys - the master key and master salt keys are binary and are two separate keys
- Base64-Encoded Keys - the master key and master salt key are combined (as specified in the *Session Description Protocol Security Descriptions for Media Streams* IETF draft) and Base64-encoded

The INIT_IPM_SECURITY_KEY inline function is provided to initialize the structure.

Field Descriptions

The fields of the IPM_SECURITY_KEY data structure are described as follows:

unSRTPLifeTime
: The lifetime of the master keys, that is, the maximum number of SRTP packets that need to be secured with the master key

unVersion
: The version of the structure

unSRTCPLifeTime
: The lifetime of the master keys, that is, the maximum number of SRTCP packets that need to be secured with the master key

unMkiLength
: The length (in bytes) of the Master Key Identifier (MKI) associated with the SRTP master key. The maximum value supported is 4. If unMkiLength is specified as 0 (zero), no MKI is attached to the SRTP packet and the MKI value is ignored.

unMkiValue
: The MKI value is the MKI that needs to be attached to the SRTP packets if the MKI length is non-zero. The maximum value is limited to 4 bytes.

unMasterKeyLength
The length (in bits) of the master key. The only value supported currently is 128.

unMasterSaltKeyLength
The length (in bits) of the master salt key. The only value supported currently is 112.

eKeyType
Identifies the type of key. Two key types are currently supported, binary and Base64-encoded keys. The eIPM_SECURITY_KEY_TYPE datatype is an enumeration that defines the supported key types as follows:

```
typedef enum
{
    IPM_SECURITY_KEYTYPE_BINARY = 1,
    IPM_SECURITY_KEYTYPE_BASE64 = 2
} eIPM_SECURITY_KEY_TYPE;
```

pvMasterKey
A void pointer to a key structure that corresponds with the key type. Currently, this is a pointer to an IPM_SECURITY_BINARY_KEY structure or an IPM_SECURITY_BASE64_KEY structure depending on the key type specified in the eKeyType field.

IPM_SESSION_INFO

```
typedef struct ipm_session_info_tag
{
    IPM_RTCP_SESSION_INFO  RtcpInfo;
    unsigned int           unQoSInfoCount;
    IPM_QOS_SESSION_INFO   QoSInfo[MAX_QOS_SESSION];

} IPM_SESSION_INFO, *PIPM_SESSION_INFO;
```

Description

This structure is a parent structure of the IPM_RTCP_SESSION_INFO and IPM_QOS_SESSION_INFO structures, and it is used by the **ipm_GetSessionInfo( )** function. It reports QoS statistics during the last IP session, including RTCP information. Note that it does not contain statistics for the current IP session.

Field Descriptions

The fields of the IPM_SESSION_INFO data structure are described as follows:

RtcpInfo
reference to RTCP session information structure IPM_RTCP_SESSION_INFO

unQoSInfoCount
number of IPM_QOS_SESSION_INFO structures to follow;
maximum sessions = MAX_QOS_SESSION

QoSInfo
reference to QoS session information structure IPM_QOS_SESSION_INFO

IPM_SRTP_PARMS

```
typedef struct ipm_srtp_parm_tag {
    unsigned int            unVersion;
    eIPM_CRYPTO_SUITE       eCryptoSuite;
    unsigned short          usKeyDerivationRate;
    unsigned short          usWindowSizeHint;
    unsigned short          usSrtpUnEncryptedFlag;
    unsigned short          usSrtcpUnEncryptedFlag;
    unsigned short          usSrtpUnAuthenticatedFlag;
    unsigned int            unSsrc;
    unsigned int            unRoc;
    unsigned int            unSeqNum;
} IPM_SRTP_PARMS, * PIPM_SRTP_PARMS;
```

Description

This structure contains parameter information for Secure RTP (SRTP). It is a child of the IPM_SECURITY_INFO structure, which in turn is a child of the IPM_MEDIA structure, which in turn is a child of the IPM_MEDIA_INFO structure used by the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions.

The INIT_IPM_SRTP_PARMS inline function is provided to initialize the structure.

The default values shown in the inline function equate to the following:

- IPM_SRTP_PARMS_VERSION
- IPM_CRYPTO_SUITE_DEFAULT (IPM_CRYPTO_AES_CM_128_HMAC_SHA1_80)
- IPM_KDR_DEFAULT (0)
- IPM_WINDOW_SIZE_HINT_DEFAULT (64)
- IPM_SRTP_UNENCRYPTED_DEFAULT (0)
- IPM_SRTCP_UNENCRYPTED_DEFAULT (0)
- IPM_SRTP_UNAUTHENTICATED_DEFAULT (0)
- IPM_SSRC_DEFAULT (0)
- IPM_ROC_DEFAULT (0)
- IPM_SEQNUM_DEFAULT (0xFFFFFFFF)

Field Descriptions

The fields of the IPM_SRTP_PARMS data structure are described as follows:

unVersion
: version of the IPM_SRTP_PARMS structure

eCryptoSuite
: Crypto suite used for authentication. Possible values are defined by the eIPM_CRYPTO_SUITE enumeration:

```
typedef enum
{
    IPM_CRYPTO_AES_CM_128_HMAC_SHA1_80 = 1,
    IPM_CRYPTO_AES_CM_128_HMAC_SHA1_32 = 2
} eIPM_CRYPTO_SUITE;
```

The enumeration values reflect the following crypto suite parameters as described in the *Session Description Protocol Security Descriptions for Media Streams* IETF draft.

Table 6. Crypto Suite Parameter Values

Characteristic	AES_CM_128_HMAC_SHA1_80	AES_CM_128_HMAC_SHA1_32
Master Key Length	128 bits	128 bits
Salt Value	112 bits	112 bits
Default Lifetime	2^{31} packets	2^{31} packets
Cipher	AES counter mode	AES counter mode
Encryption Key	128 bits	128 bits
MAC	HMAC-SHA1	HMAC-SHA1
Authentication Tag	80 bits	32 bits
SRTP Auth Key Length	160 bits	160 bits
SRTCP Auth Key Length	160 bits	160 bits

usKeyDerivationRate
: defines the Key Derivation Rate (KDR), that is the rate at which session keys are derived from the master key and master salt key. If KDR is set to 0 (zero, the default), the key is derived only once. This field takes an integer value in the range 1 to 24, which corresponds to a KDR value in the range 2^1 to 2^{24}.

usWindowSizeHint
: the SRTP_WINDOW_SIZE parameter that protects against replay attacks (that is, the capturing of a packet and later reinsertion into a stream). The default value (and minimum value) is 64.

usSrtpUnEncryptedFlag
: flag for enabling or disabling the encryption of SRTP packets or the use of the NULL cipher in SRTP. Possible values are 1 (SRTP messages are **not** encrypted) or 0 (SRTP messages are encrypted). The default value is 0.

usSrtcpUnEncryptedFlag
: flag for enabling or disabling the encryption of SRTCP packets or the use of the NULL cipher in SRTCP. Possible values are 1 (SRTCP messages are not encrypted) or 0 (SRTCP messages are encrypted). The default value is 0.

usSrtpUnAuthenticatedFlag
: flag for enabling or disabling SRTP authentication. Possible values are 1 (SRTP messages are not authenticated) or 0 (SRTP messages are authenticated). The default value is 0.

unSsrc
: specifies how the Synchronized Source (SSRC) identifier is obtained. Possible values are:

- 0, obtain the SSRC that is either automatically generated by HMP software for outgoing RTP packets or taken from inbound SRTP packets (default)
- 1, obtain by negotiation via SDP (passed to IP Media library)

Note: The unSsrc field is not currently supported.

unRoc

specifies how the Rollover Counter (ROC) is obtained. Possible values are:

- 0, estimate the ROC from SRTP packets (default)
- 1, obtain by negotiation via SDP (passed to IP Media library)

Note: The unRoc field is not currently supported.

unSeqNum

specifies the starting sequence number for SRTP packets. By default, the sequence number is taken from the SRTP packet. If the starting sequence number is negotiated, the same value can be passed to the IP Media library. The default is 0xFFFFFFFF.

Note: The unSeqNum field is not currently supported.

IPM_VIDEO_CODER_INFO

```
typedef struct ipm_video_coder_info_tag
{
   unsigned int            unVersion;           /* structure version                    */
   eIPM_CODER_TYPE         eCoderType;          /* The coder Type                       */
   unsigned int            unFrameRate;         /* Frame rate, fps x100                 */
   unsigned int            unSamplingRate;      /* Sampling rate                        */
   unsigned int            unCoderPayloadType;  /* Type of coder payload supported      */

} IPM_VIDEO_CODER_INFO, *PIPM_VIDEO_CODER_INFO;
```

Description

This structure contains the coder properties that will be used in a video IP session. IPM_VIDEO_CODER_INFO is a child of the IPM_MEDIA union, which is a child of the IPM_MEDIA_INFO structure that is used by the **ipm_GetLocalMediaInfo( )**, **ipm_ModifyMedia( )**, and **ipm_StartMedia( )** functions.

Field Descriptions

The fields of the IPM_VIDEO_CODER_INFO data structure are described as follows.

unVersion
: version number of the data structure. Applications should *not* manipulate this field.

eCoderType
: type of coder to be used for streaming media operations. The following values are defined:
  - CODER_TYPE_H263 – H.263 video coder

unFrameRate
: video frame rate specified as 100x the actual number of frames per second. The following values are valid:
  - 600 – frame rate is 6 fps
  - 1000 – frame rate is 10 fps
  - 1500 – frame rate is 15 fps

unSamplingRate
: sampling rate of media stream. This value is typically set to 90,000.

unCoderPayloadType
: RTP header payload type using RFC 1890 standard definitions. The application is responsible for negotiating this value between the two endpoints. This may be set to any value for non-standard coders or if the application does not require interoperability with third-party applications. Values: 0 to 127; 96 to 127 is the dynamic payload range.

Error Codes 5

This chapter describes the error/cause codes supported by the Dialogic® IP Media software error library, *ipmerror.h*. All Dialogic® IP Media library functions return a value that indicates the success or failure of the function call. Success is indicated by a return value of zero or a non-negative number. Failure is indicated by a value of -1.

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

If an error occurs during execution of an asynchronous function, the IPMEV_ERROR event is sent to the application. No change of state is triggered by this event. Upon receiving the IPMEV_ERROR event, the application can retrieve the reason for the failure using the SRL functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )**.

The IP Media software error library contains the following error codes, listed in alphabetical order. The list also identifies the functions that may return the particular error code.

EIPM_BADPARM
Bad argument or parameter. All IP Media library functions except **ipm_Open( )**.

EIPM_BUSY
Device busy. **ipm_InitResponseSend( )ipm_InitSend( )ipm_StartMedia( )**

EIPM_CONFIG
Configuration error. **ipm_Close( )**

EIPM_EVT_EXIST
Event already enabled. **ipm_EnableEvents( )**

EIPM_EVT_LIST_FULL
Too many events. **ipm_EnableEvents( )**

EIPM_FWERROR
Firmware error. **ipm_Close( )**, **ipm_GetParm( )**, **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_SetParm( )**, **ipm_Stop( )**, **ipm_UnListen( )**

EIPM_INTERNAL
Internal error. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_InitResponseSend( )**, **ipm_InitSend( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_INTERNAL_INIT
Internal initialization error.

EIPM_INV_DEVNAME
Invalid device name.

EIPM_INV_EVT
Invalid event. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**

EIPM_INV_MODE
Invalid mode. **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_InitResponseSend( )**, **ipm_InitSend( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_StartMedia( )**

EIPM_INV_STATE
Invalid state. Error indicates that initial command did not complete before another function call was made. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_InitResponseSend( )**, **ipm_InitSend( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_NOERROR
No error.

EIPM_NOMEMORY
Memory allocation error.

EIPM_RESOURCEINUSE
Resource in use or not available.

EIPM_SRL
SRL error.

EIPM_SRL_SYNC_TIMEOUT
SRL timeout.

EIPM_SYSTEM
System error. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_InitResponseSend( )**, **ipm_InitSend( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_TIMEOUT
Timeout.

EIPM_UNSUPPORTED
Function unsupported. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**

Glossary

Codec: see COder/DECoder

COder/DECoder: A circuit used to convert analog voice data to digital and digital voice data to analog audio.

Computer Telephony (CT): Adding computer intelligence to the making, receiving, and managing of telephone calls.

DTMF: Dual-Tone Multi-Frequency

Dual-Tone Multi-Frequency: A way of signaling consisting of a push-button or touch-tone dial that sends out a sound consisting of two discrete tones that are picked up and interpreted by telephone switches (either PBXs or central offices).

Emitting Gateway: called by a G3FE. It initiates IFT service for the calling G3FE and connects to a Receiving Gateway.

E1: The 2.048 Mbps digital carrier system common in Europe.

FCD file: An ASCII file that lists any non-default parameter settings that are necessary to configure a DM3 hardware/firmware product for a particular feature set. The downloader utility reads this file, and for each parameter listed generates and sends the DM3 message necessary to set that parameter value.

Frame: A set of SCbus/CT Bus timeslots which are grouped together for synchronization purposes. The period of a frame is fixed (at 125 µsec) so that the number of time slots per frame depends on the SCbus/CT Bus data rate.

G3FE: Group 3 Fax Equipment. A traditional fax machine with analog PSTN interface.

Gatekeeper: An H.323 entity on the Internet that provides address translation and control access to the network for H.323 Terminals and Gateways. The Gatekeeper may also provide other services to the H.323 terminals and Gateways, such as bandwidth management and locating Gateways.

Gateway: A device that converts data into the IP protocol. It often refers to a voice-to-IP device that converts an analog voice stream, or a digitized version of the voice, into IP packets.

H.323: A set of International Telecommunication Union (ITU) standards that define a framework for the transmission of real-time voice communications through Internet protocol (IP)-based packet-switched networks. The H.323 standards define a gateway and a gatekeeper for customers who need their existing IP networks to support voice communications.

IAF: Internet Aware Fax. The combination of a G3FE and a T.38 gateway.

IFP: Internet Facsimile Protocol

IFT: Internet Facsimile Transfer

International Telecommunications Union (ITU): An organization established by the United Nations to set telecommunications standards, allocate frequencies to various uses, and hold trade shows every four years.

Internet: An inter-network of networks interconnected by bridges or routers. LANs described in H.323 may be considered part of such inter-networks.

Internet Protocol (IP): The network layer protocol of the transmission control protocol/Internet protocol (TCP/IP) suite. Defined in STD 5, Request for Comments (RFC) 791. It is a connectionless, best-effort packet switching protocol.

Internet Service Provider (ISP): A vendor who provides direct access to the Internet.

Internet Telephony: The transmission of voice over an Internet Protocol (IP) network. Also called Voice over IP (VoIP), IP telephony enables users to make telephone calls over the Internet, intranets, or private Local Area Networks (LANs) and Wide Area Networks (WANs) that use the Transmission Control Protocol/Internet Protocol (TCP/IP).

ITU: See International Telecommunications Union.

Jitter: The deviation of a transmission signal in time or phase. It can introduce errors and loss of synchronization in high-speed synchronous communications.

NIC (Network Interface Card): Adapter card inserted into computer that contains necessary software and electronics to enable a station to communicate over network.

PCD file: An ASCII text file that contains product or platform configuration description information that is used by the DM3 downloader utility program. Each of these files identifies the hardware configuration and firmware modules that make up a specific hardware/firmware product. Each type of DM3-based product used in a system requires a product-specific PCD file.

PSTN: see Public Switched Telephone Network

Public Switched Telephone Network: The telecommunications network commonly accessed by standard telephones, key systems, Private Branch Exchange (PBX) trunks and data equipment.

Reliable Channel: A transport connection used for reliable transmission of an information stream from its source to one or more destinations.

Reliable Transmission: Transmission of messages from a sender to a receiver using connection-mode data transmission. The transmission service guarantees sequenced, error-free, flow-controlled transmission of messages to the receiver for the duration of the transport connection.

RTCP: Real Time Control Protocol

RTP: Real Time Protocol

SIP: Session Initiation Protocol: an Internet standard specified by the Internet Engineering Task Force (IETF) in RFC 3261. SIP is used to initiate, manage, and terminate interactive sessions between one or more users on the Internet.

T1: A digital transmission link with a capacity of 1.544 Mbps used in North America. Typically channeled into 24 digital subscriber level zeros (DS0s), each capable of carrying a single voice conversation or data stream. T1 uses two pairs of twisted pair wires.

TCP: see Transmission Control Protocol

Terminal: An H.323 Terminal is an endpoint on the local area network which provides for real-time, two-way communications with another H.323 terminal, Gateway, or Multipoint Control Unit. This communication consists of control, indications, audio, moving color video pictures, and/or data between the two terminals. A terminal may provide speech only, speech and data, speech and video, or speech, data, and video.

Transmission Control Protocol: The TCP/IP standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection.

UDP: see User Datagram Protocol

UDPTL: Facsimile UDP Transport Layer protocol

User Datagram Protocol: The TCP/IP standard protocol that allows an application program on one machine to send a datagram to an application program on another machine. Conceptually, the important difference between UDP datagrams and IP datagrams is that UDP includes a protocol port number, allowing the sender to distinguish among multiple destinations on the remote machine.

VAD: Voice Activity Detection

Index

C

D

F

I

M

N

P

Q

R

S

T

V