



# IP Media Library API for Linux and Windows

Library Reference

---

*May 2006*



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This IP Media Library API for Linux and Windows Library Reference as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Copyright © 2001-2006 Intel Corporation. All Rights Reserved.

Dialogic, Intel, Intel logo, and Intel NetStructure are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other names and brands may be claimed as the property of others.

Publication Date: May 2006

Document Number: 05-1833-006

Intel  
1515 Route 10  
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:  
<http://developer.intel.com/design/telecom/support>

For **Products and Services Information**, visit the Intel Telecom and Compute Products website at:  
<http://www.intel.com/design/network/products/telecom>

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:  
<http://www.intel.com/buy/networking/telecom.htm>



# Contents

---

	<b>Revision History</b> .....	6
	<b>About This Publication</b> .....	11
	Purpose .....	11
	Intended Audience .....	11
	How to Use This Publication .....	11
	Related Information .....	12
<b>1</b>	<b>Function Summary by Category</b> .....	13
1.1	System Control Functions .....	13
1.2	I/O (Input/Output) Functions .....	14
1.3	Media Session Functions .....	14
1.4	Quality of Service (QoS) Functions .....	14
1.5	IP Media Function Support by Platform .....	15
<b>2</b>	<b>Function Information</b> .....	17
2.1	Function Syntax Conventions .....	17
	ipm_Close() – close an IP channel device .....	18
	ipm_DisableEvents() – disable IP notification events .....	20
	ipm_EnableEvents() – enable IP notification events .....	24
	ipm_GetCTInfo() – return information about a voice channel of an IPM device .....	28
	ipm_GetLocalMediaInfo() – retrieve properties for the local media channel .....	30
	ipm_GetParm() – retrieve the current value of a parameter .....	34
	ipm_GetQoSAlarmStatus() – retrieve ON/OFF state of QoS alarms .....	37
	ipm_GetQoSThreshold() – retrieve QoS alarm threshold settings .....	40
	ipm_GetSessionInfo() – retrieve statistics for a session .....	44
	ipm_GetXmitSlot() – return TDM time slot information for an IP channel .....	48
	ipm_Listen() – connect an IP channel to a TDM time slot .....	51
	ipm_ModifyMedia() – modify properties of active media session .....	55
	ipm_Open() – open an IP channel device .....	59
	ipm_Ping() – generate a “ping” message to a remote IP address .....	62
	ipm_ReceiveDigits() – enable the IP channel to receive digits .....	65
	ipm_ResetQoSAlarmStatus() – reset QoS alarm(s) to the OFF state .....	69
	ipm_SendDigits() – generate digits to the TDM bus .....	72
	ipm_SetParm() – set value for specified parameter .....	75
	ipm_SetQoSThreshold() – change QoS alarm threshold settings .....	78
	ipm_SetRemoteMediaInfo() – set media properties and starts the session .....	81
	ipm_StartMedia() – set media properties and start the session .....	85
	ipm_Stop() – stop operations on the specified IP channel .....	89
	ipm_UnListen() – stop listening to the TDM time slot .....	92
<b>3</b>	<b>Events</b> .....	95
<b>4</b>	<b>Data Structures</b> .....	99

CT_DEVINFO – channel/time slot device information . . . . .	100
IPM_CLOSE_INFO – reserved for future use . . . . .	102
IPM_CODER_INFO – coder properties used in an IP session . . . . .	103
IPM_DIGIT_INFO – used to transfer digits over IP network and TDM bus . . . . .	107
IPM_EVENT_INFO – used for IP event notification . . . . .	108
IPM_FAX_SIGNAL – detected tone information definition . . . . .	109
IPM_MEDIA – parent of port and coder info structures . . . . .	110
IPM_MEDIA_INFO – parent of IP_MEDIA, contains session info . . . . .	111
IPM_OPEN_INFO – reserved for future use . . . . .	112
IPM_PARM_INFO – used to set or retrieve parameters for an IP channel . . . . .	113
IPM_PING_INFO – ping response information . . . . .	115
IPM_PING_PARM – ping parameter information . . . . .	116
IPM_PORT_INFO – RTP and RTCP port properties . . . . .	117
IPM_QOS_ALARM_DATA – data associated with QoS alarms . . . . .	118
IPM_QOS_ALARM_STATUS – parent of QoS alarm data, contains alarm status . . . . .	119
IPM_QOS_SESSION_INFO – QoS statistics for an IP session . . . . .	120
IPM_QOS_THRESHOLD_DATA – QoS alarm threshold settings for an IP channel . . . . .	121
IPM_QOS_THRESHOLD_INFO – parent of threshold data structures . . . . .	124
IPM_RFC2833_SIGNALID_INFO – RFC 2833 signal ID and state info . . . . .	125
IPM_RTCP_SESSION_INFO – session information for RTCP . . . . .	126
IPM_SESSION_INFO – parent structure containing RTCP and QoS info . . . . .	128
SC_TSINFO – TDM bus (CT Bus) time slot information . . . . .	129
<b>5 Error Codes . . . . .</b>	<b>131</b>
<b>Glossary . . . . .</b>	<b>133</b>
<b>Index . . . . .</b>	<b>137</b>

## Tables

---

1	IP Media Function Support by Platform . . . . .	15
2	Supported Coder Properties for Intel NetStructure IPT Series Boards . . . . .	104
3	Supported Coder Properties for Intel NetStructure DM/IP Series Boards . . . . .	106
4	eIPM_PARM Parameters and Values . . . . .	113
5	Quality of Service Parameter Defaults for IPT Boards . . . . .	122
6	Quality of Service Parameter Defaults for DM/IP Boards . . . . .	122
7	eIPM_RFC2833_SIGNAL_ID Values . . . . .	125

## Revision History

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

Document No.	Publication Date	Description of Revisions
05-1833-006	May 2006	<p>General: Updated specific eMedia types to use MEDIATYPE_AUDIO_xxx equivalents.</p> <p><a href="#">Related Information</a> section: Added hyperlinks for RFCs mentioned in document.</p> <p><a href="#">ipm_GetLocalMediaInfo( )</a> function: Updated description to clarify eMediaType and unCount as members of data structures referenced by the pMediaInfo function parameter and enumerate the allowed values.</p> <p><a href="#">ipm_ModifyMedia( )</a> function: Updated the list of properties that can be modified; also updated the code example. Changed support to DM/IP boards only.</p> <p><a href="#">ipm_StartMedia( )</a> function: Updated eDirection value descriptions.</p> <p><a href="#">IPM_CODER_INFO</a> structure: In the unCoderPayloadType field description, added note to clarify the use of values in the dynamic range.</p> <p><a href="#">IPM_RTCP_SESSION_INFO</a> structure: Updated the note to indicate partial support for IPT boards.</p> <p><a href="#">IPM_SESSION_INFO</a> structure: Updated the note to indicate partial support for IPT boards.</p> <p><a href="#">Events</a> chapter: Updated the IPMEV_QOS_ALARM event description.</p>
05-1833-005	August 2005	<p><a href="#">ipm_DisableEvents( )</a> function: Noted that EVT_LOSTPACKETS is only supported for IPT series boards. Added EVT_NETWORKFAILURE event.</p> <p><a href="#">ipm_EnableEvents( )</a> function: Noted that EVT_LOSTPACKETS is only supported for IPT series boards. Added EVT_NETWORKFAILURE event.</p> <p><a href="#">ipm_Listen( )</a> function: Added caution regarding sharing of timeslots (PTR#33174)</p> <p><a href="#">ipm_ModifyMedia( )</a>: New function</p> <p><a href="#">ipm_ReceiveDigits( )</a> function: Updated caution about RTP session requirement</p> <p><a href="#">ipm_SendDigits( )</a> function: Updated caution about RTP session requirement</p> <p><a href="#">ipm_SendRFC2388SignalIDToIP</a> function: Removed as unsupported</p> <p><a href="#">ipm_SetRemoteMediaInfo( )</a> function: Noted that half-duplex modes are supported for IPT boards only. Noted that multicast server mode is supported for DM/IP boards only. Removed define for unsupported multicast client mode</p> <p><a href="#">ipm_StartMedia( )</a> function: Noted that half-duplex modes are supported for IPT boards only. Noted that multicast server mode is supported for DM/IP boards only. Removed define for unsupported multicast client mode</p> <p><a href="#">ipm_UnListen( )</a> function: Added caution regarding sharing of timeslots (PTR#33174)</p> <p>Events chapter: Removed unsupported IPMEV_RFC2833SIGNALRECEIVED event</p> <p><a href="#">IPM_CODER_INFO</a> data structure: Added defines and table entries for GSM AMR-NB coder. Added note to unCoderPayloadType description (PTR#33921). Deleted unsupported 5ms frame size value for G.711 coder.</p> <p><a href="#">IPM_PARM_INFO</a> data structure: Added info about using PARMCH_TOS for DiffServ field (DSCP)</p>

Document No.	Publication Date	Description of Revisions
05-1833-005 (continued)	August 2005	<p>IPM_QOS_ALARM_DATA data structure: Noted that QOSTYPE_LOSTPACKETS is only supported for IPT series boards. Added QOSTYPE_NETWORKFAILURE alarm type.</p> <p>IPM_QOS_SESSION_INFO data structure: Noted that QOSTYPE_LOSTPACKETS is only supported for IPT series boards</p> <p>IPM_QOS_THRESHOLD_DATA data structure: Added information on granularity of parameters (PTR#35692). Noted that QOSTYPE_LOSTPACKETS is only supported for IPT series boards.</p>
05-1833-004	September 2004	<p>ipm_DisableEvents( ) function: Removed unsupported EVT_DTMFDISCARDED event</p> <p>ipm_EnableEvents( ) function: Removed unsupported EVT_DTMFDISCARDED event</p> <p>ipm_GetLocalMediaInfo( ) function: Corrected MEDIATYPE types referred to in the Cautions and Code Example sections</p> <p>ipm_GetQoSAlarmStatus( ) function: Corrected code example</p> <p>ipm_GetSessionInfo( ) function: Added info on NULL pointer in async mode</p> <p>ipm_GetXmitSlot( ) function: Added info on NULL pointer in async mode</p> <p>ipm_Open( ) function: Removed caution about sync mode only for DM/IP boards.</p> <p>ipm_Ping( ) function: Corrected code example</p> <p>ipm_ReceiveDigits( ) function: Clarified usage of data structure. Added caution regarding active RTP session requirement for receiving digits.</p> <p>ipm_SendDigits( ) function: Added info on maximum number of digits. Added caution regarding active RTP session requirement for sending digits.</p> <p>ipm_SetQoSThreshold( ) function: Revised code example to use correct minimum value (100) for unTimeInterval</p> <p>ipm_SetRemoteMediaInfo( ) function: Documented function as deprecated. Corrected code example.</p> <p>ipm_StartMedia( ) function: Added caution to avoid setting IP address 0.0.0.0 (PTR#32986). Corrected name of completion event. Corrected code example.</p> <p>ipm_Stop( ) function: Removed two unsupported eIPM_STOP_OPERATION values</p> <p>CT_DEVINFO data structure: Removed info irrelevant to structure's use in IPML</p> <p>IPM_CODER_INFO data structure: Updated tables of supported coders</p> <p>IPM_DIGIT_INFO data structure: Added info about maximum number of digits and send vs. receive usage differences</p> <p>IPM_PARM_INFO structure: Removed unsupported PARMCH_RFC2833REDLEVEL parameter values. Expanded and corrected descriptions of all parameters.</p> <p>IPM_PORT_INFO data structure: Added caution to avoid IP address 0.0.0.0 (PTR#32986).</p> <p>IPM_QOS_SESSION_INFO data structure: Corrected names of QoS type enums.</p> <p>IPM_QOS_THRESHOLD_DATA data structure: Corrected names of QoS type defines. Added value ranges to description of unFaultThreshold. Expanded and corrected descriptions of all fields, including default values.</p>

Document No.	Publication Date	Description of Revisions
05-1833-003	November 2003	<p>ipm_DisableEvents( ): Removed unsupported EVT_T38FAXTONE event</p> <p>ipm_EnableEvents( ): Removed unsupported EVT_T38FAXTONE event</p> <p>ipm_GetCTInfo( ): New function</p> <p>ipm_SetRemoteMediaInfo( ) function: corrected example</p> <p>ipm_StartMedia( ) function: corrected code example</p> <p>CT_DEVINFO structure: added page</p> <p>IPM_CODER_INFO: Corrected G.726 coder entry for VAD to not supported</p> <p>IPM_PARM_INFO data structure: Added valid values and variable type for PARMCH_RFC2833EVT_TX_PLT and PARMCH_RFC2833EVT_RX_PLT. Deleted PARMCH_RFC2833MUTE_AUDIO, PARMCH_RFC2833TONE_TX_PLT, and PARMCH_RFC2833TONE_RX_PLT</p> <p>Error Codes chapter: added EIPM_RESOURCEINUSE</p> <p>Function Summary by Category chapter: New IP Media Function Support by Platform section with new table</p> <p>ipm_GetLocalMediaInfo( ) function: Minor changes in code example</p> <p>ipm_GetParm( ) function: Updated code example</p> <p>ipm_GetQoSAlarmStatus( ) function: Added note regarding support on IPT Series boards. Minor changes in code example.</p> <p>ipm_GetQoSThreshold( ) function: Revised description for pQoSThresholdInfo parameter. Multiple changes in code example.</p> <p>ipm_GetSessionInfo( ) function: Added note and caution regarding support on IPT Series boards. Minor changes in code example.</p> <p>ipm_GetXmitSlot( ) function: Modified event handler in code example</p> <p>ipm_Listen( ) function: Multiple minor changes in code example</p> <p>ipm_Open( ) function: Added caution regarding separate processes</p> <p>ipm_Ping( ) function reference: Added HMP to non-supporting platforms. Minor changes in code example.:</p> <p>ipm_ReceiveDigits( ) function reference: New caution regarding out-of-band vs. DTMF modes</p> <p>ipm_ResetQoSAlarmStatus( ) function reference: New note about function not being supported on IPT Series boards</p>
05-1833-002 (continued)	November 2002	<p>ipm_SendDigits( ) function reference: New note about function not being supported on Host Media Processing software</p> <p>Minor changes in code example</p> <p>ipm_SendRFC2833SignalIDToIP( ) function reference: New code example</p> <p>ipm_SetParm( ) function reference: Minor changes in code example</p> <p>ipm_SetRemoteMediaInfo( ) function reference: Minor changes in code example</p> <p>ipm_StartMedia( ) function reference: Minor changes in code example</p> <p>IPM_RFC2833_SIGNALID_INFO data structure reference: Corrected statement about default value for eState.</p> <p>ipm_Stop( ) function reference: Changed description of IPMEV_STOPPED event</p> <p>Deleted caution regarding automatic call to ipm_UnListen( )</p> <p>ipm_UnListen( ) function reference: Softened caution regarding synchronous and asynchronous modes</p> <p>IPM_DIGIT_INFO data structure reference: Corrected data type in description of eDigitType field</p>



Document No.	Publication Date	Description of Revisions
05-1833-002	November 2002	<p>IPM_PARM_INFO data structure reference: Revised defines and descriptions for eIPM_PARM values</p> <p>IPM_PORT_INFO data structure reference: Corrected name of cIPAddress[IP_ADDR_SIZE]</p> <p>IPM_RFC2833_SIGNALID_INFO data structure reference: New note about structure not being supported on IPT Series boards. Revised and expanded descriptions of values for eStat:</p>
05-1833-001	September 2002	Initial version of this document





# About This Publication

---

The following topics provide information about this publication:

- [Purpose](#)
- [Intended Audience](#)
- [How to Use This Publication](#)
- [Related Information](#)

## Purpose

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

This guide specifically applies to the IP Media Library that is provided with the Intel® Dialogic® System Release software for use with Intel NetStructure® IPT Series and Intel NetStructure® DM/IP Series boards. The IP Media Library API as implemented in the Intel NetStructure® Host Media Processing Software has some significant functional differences, and separate versions of the IP Media Library documents are provided with the software release for that product.

## Intended Audience

This guide is intended for software developers who will access the IP media software. 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 IP media software. This publication assumes that you are familiar with the Linux\* or Windows\* operating system and the C programming language. It is helpful to keep the *Voice API Library Reference* handy as you develop your application.

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 IP media software.
- [Chapter 4, “Data Structures”](#) provides details about each data structure used by the IP media software, including fields and descriptions.
- [Chapter 5, “Error Codes”](#) lists the error codes included in the IP media software.

## Related Information

The following guides may also be used to develop IP technology-based applications:

- *IP Media Library API Programming Guide*
- *Global Call IP Technology Guide*
- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *Standard Runtime Library API for Linux and Windows Operating Systems Library Reference*
- Internet Engineering Task Force (IETF) Request for Comments RFC 2198, *RTP Payload for Redundant Audio Data*, <http://www.ietf.org/rfc/rfc2198.txt>
- Internet Engineering Task Force (IETF) Request for Comments RFC 2833, *RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals*, <http://www.ietf.org/rfc/rfc2833.txt>
- Internet Engineering Task Force (IETF) Request for Comments RFC 3551, *RTP Profile for Audio and Video Conferences with Minimal Control*, <http://www.ietf.org/rfc/rfc3551.txt?number=3551>
- <http://developer.intel.com/design/telecom/support/> (for technical support)
- <http://www.intel.com/design/network/products/telecom/> (for product information)

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

- [System Control Functions . . . . .](#) 13
- [I/O \(Input/Output\) Functions . . . . .](#) 14
- [Media Session Functions . . . . .](#) 14
- [Quality of Service \(QoS\) Functions . . . . .](#) 14
- [IP Media Function Support by Platform . . . . .](#) 15

## 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\\_Ping\(\)\*\*](#)

generates a message to a remote IP address

### [\*\*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\_ModifyMedia()**

modifies the properties of an active media session

### **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 IP Media Function Support by Platform

Table 1, “IP Media Function Support by Platform”, on page 15 provides an alphabetical listing of IP media API functions. The table indicates which platforms are supported for each of the functions

Intel NetStructure® DM/IP Series boards

These boards feature 24 to 60 ports-per-slot of both public network and Internet connectivity plus on-board voice, fax, and speech processing. The boards are scalable to support access gateways, IP-PBXs, and media server applications.

Intel NetStructure® IPT Series boards

These boards provide a high-density, standards-based VoIP interface for developing scalable, carrier-grade IP telephony gateways and media servers.

Although a function may be supported on all the platforms, there may be some restrictions on its use. For example, some parameters or parameter values may not be supported. For details, see the function reference descriptions in Chapter 2, “Function Information”.

**Table 1. IP Media Function Support by Platform**

Function	DM/IP Boards	IPT Boards
<a href="#">ipm_Close( )</a>	S	S
<a href="#">ipm_DisableEvents( )</a>	S	S
<a href="#">ipm_EnableEvents( )</a>	S	S
<a href="#">ipm_GetCTInfo( )</a>	S	NS
<a href="#">ipm_GetLocalMediaInfo( )</a>	S	S
<a href="#">ipm_GetParm( )</a>	S	S
<a href="#">ipm_GetQoSAlarmStatus( )</a>	S	NS
<a href="#">ipm_GetQoSThreshold( )</a>	S	S†
<a href="#">ipm_GetSessionInfo( )</a>	S	NS
<a href="#">ipm_GetXmitSlot( )</a>	S	S
<a href="#">ipm_Listen( )</a>	S	S
<a href="#">ipm_ModifyMedia( )</a>	S†	NS
<a href="#">ipm_Open( )</a>	S	S
<a href="#">ipm_Ping( )</a>	NS	S
<a href="#">ipm_ReceiveDigits( )</a>	S	S
<a href="#">ipm_ResetQoSAlarmStatus( )</a>	S	NS
<a href="#">ipm_SendDigits( )</a>	S	S
<a href="#">ipm_SetParm( )</a>	S	S
<a href="#">ipm_SetQoSThreshold( )</a>	S	S†
<b>Legend:</b> NS = Not Supported, S = Supported, † = Variance between platforms, refer to Function Description for more information.		

Table 1. IP Media Function Support by Platform (Continued)

Function	DM/IP Boards	IPT Boards
<a href="#">ipm_SetRemoteMediaInfo( )</a>	S (deprecated)	S (deprecated)
<a href="#">ipm_StartMedia( )</a>	S	S
<a href="#">ipm_Stop( )</a>	S	S
<a href="#">ipm_UnListen( )</a>	S	S
<b>Legend:</b> NS = Not Supported, S = Supported, † = Variance between platforms, refer to Function Description for more information.		



This chapter contains a detailed description of each IP Media library (IPML) function, presented in alphabetical order.

## 2.1 Function Syntax Conventions

The IP Media library (IPML) 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:** srlib.h  
ipmlib.h

**Category:** System Control

**Mode:** synchronous only

**Platform:** DM/IP, IPT

### ■ Description

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

Parameter	Description
<b>nDeviceHandle</b>	IP Media device handle returned by <b>ipm_Open( )</b>
<b>pCloseInfo</b>	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\_BADPARAM  
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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pEvents</b>	<p>array of enumerations that specifies the events to disable</p> <p>The eIPM_EVENT data type is an enumeration that defines the following values:</p> <ul style="list-style-type: none"> <li>• EVT_LOSTPACKETS – channel-level QoS alarm for excessive percentage of lost packets (IPT boards only)</li> <li>• EVT_JITTER – channel-level QoS alarm for excessive average jitter</li> <li>• EVT_NETWORKFAILURE – board-level QoS alarm for RTP network disruption (IPT boards only); note that the function call will fail if this event type is specified for a line device handle rather than a board device handle</li> <li>• EVT_ROUNDTRIPLATENCY – channel-level QoS alarm for RTP packet latency (IPT boards only)</li> <li>• EVT_RFC2833 – channel-level RFC 2833 event</li> </ul>

Parameter	Description
<b>unNumOfEvents</b>	number of events to disable (number of enumerations in <b>pEvents</b> array)
<b>usMode</b>	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 when disabling EVT\_NETWORKFAILURE if **nDeviceHandle** specifies a channel device rather than a board device, or when the board device is not an Intel NetStructure® IPT Series board.
- The function fails if any event other than EVT\_NETWORKFAILURE is specified when **nDeviceHandle** specifies a board device rather than a channel device.

### ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

#### EIPM\_BADPARAM

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_enbhdr( 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_getevtttype();
    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:** DM/IP, IPT

### ■ 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.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pEvents</b>	<p>array of enumerations that specifies the events to enable</p> <p>The eIPM_EVENT data type is an enumeration that defines the following values:</p> <ul style="list-style-type: none"> <li>• EVT_LOSTPACKETS – channel-level QoS alarm for excessive percentage of lost packets (IPT boards only)</li> <li>• EVT_JITTER – channel-level QoS alarm for excessive average jitter</li> <li>• EVT_NETWORKFAILURE – board-level QoS alarm for RTP network disruption (IPT boards only); note that the function call will fail if this event type is specified for a line device handle rather than a board device handle</li> <li>• EVT_ROUNDTRIPLATENCY – channel-level QoS alarm for RTP packet latency (IPT boards only)</li> <li>• EVT_RFC2833 – channel-level RFC 2833 event</li> </ul>



Parameter	Description
<b>unNumOfEvents</b>	number of events to enable (number of enumerations in <b>pEvents</b> array)
<b>usMode</b>	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 when enabling EVT\_NETWORKFAILURE if **nDeviceHandle** specifies a channel device rather than a board device, or when the board device is not an Intel NetStructure® IPT Series board.
- The function fails if any event other than EVT\_NETWORKFAILURE is specified when **nDeviceHandle** specifies a board device rather than a channel device.

#### ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR( )** and **ATDV\_ERRMSGP( )** to return one of the following errors:

##### EIPM\_BADPARAM

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_enbhdlnr( 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_getevtttype();
    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\_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:** DM/IP

### ■ 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.

**Note:** This function is not supported on Intel NetStructure IPT Series boards.

Parameter	Description
<b>nDeviceHandle</b>	specifies the valid IP channel handle obtained when the channel was opened using <b>ipm_Open( )</b>
<b>pCTInfo</b>	specifies a pointer to the CT_DEVINFO structure that contains the IP channel device information
<b>usMode</b>	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\_BADPARAM  
Invalid parameter

### ■ Example

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



*return information about a voice channel of an IPM device — `ipm_GetCTInfo()`*

```
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_ERRMSGF(chdev));
        exit(1);
    }

    printf("%s Product Id = 0x%x, Family = %d, Mode = %d, Network = %d, Bus
           mode = %d, Encoding = %d", ATDV_NAMEF(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:** DM/IP, IPT

### ■ Description

The **ipm\_GetLocalMediaInfo( )** function retrieves properties for the local media channel. This function retrieves the local IP address and port information for the 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 for which port information is to be retrieved. The allowed combinations are as follows:

- 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.

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

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 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

Parameter	Description
<b>pMediaInfo</b>	pointer to structure that contains local IP address and IP channel port information for RTP / RTCP ports or T.38 port  See the <a href="#">IPM_MEDIA_INFO</a> data structure page for details.
<b>usMode</b>	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 SRL functions to retrieve IPM\_MEDIA\_INFO structure fields.

### IPMEV\_ERROR

Indicates that the function failed.

## ■ Cautions

- When using Intel NetStructure IPT boards, the following limitations apply:
  - For a non-load balancing configuration, if this function is called multiple times, it could return a different port number for a specified channel.
  - In load-balancing mode, if this function is called multiple times, it could return a different IP/Port pair each time.
- When using Intel NetStructure DMIP products, if the **ipm\_GetLocalMediaInfo()** function is used for the retrieval of T.38 fax ports and is followed by a call to **ipm\_StartMedia()**, a T.38 fax session is automatically started. A subsequently call to **ipm\_GetLocalMediaInfo()** with RTP specified results in a switch to an RTP session.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

### EIPM\_BADPARAM

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 = 1;
    // 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++)
        {
            if(MEDIATYPE_AUDIO_LOCAL_RTP_INFO == pMediaInfo->MediaData[i].eMediaType)
                printf("MediaType = MEDIATYPE_RTP_INFO!!\n");
            printf("PortId= %d\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
            printf("IPAddress=%s\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.cIPAddress);
        }
        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:** DM/IP, IPT

### ■ 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 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
<b>nDeviceHandle</b>	handle of the IP media device
<b>*pParmInfo</b>	pointer to structure that contains IP channel parameter values See the <a href="#">IPM_PARM_INFO</a> data structure page for details.
<b>usMode</b>	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 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\_BADPARAM**  
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_getevtdata();
    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:** srlib.h  
ipmlib.h

**Category:** QoS

**Mode:** asynchronous or synchronous

**Platform:** DM/IP

### ■ 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.

**Note:** This function is not supported on Intel NetStructure® IPT boards.

Use **ipm\_ResetQoSAlarmStatus()** to reset the QoS alarm state.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media channel device
<b>pQoSAlarmInfo</b>	pointer to structure that contains alarm identifier and alarm status values See <a href="#">IPM_QOS_ALARM_STATUS</a> for details.
<b>usMode</b>	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\_BADPARAM**

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_getevtttype();
    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:** DM/IP, IPT

### ■ Description

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

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pQoSThresholdInfo</b>	pointer to <a href="#">IPM_QOS_THRESHOLD_INFO</a> structure which contains one or more <a href="#">IPM_QOS_THRESHOLD_DATA</a> structures
<b>usMode</b>	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 (such as the network failure alarm for Intel NetStructure® boards) 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\_BADPARAM**

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_enbhdr( 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_getevtttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdata();
    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:** DM/IP

### ■ 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.

**Note:** This function is not supported on Intel NetStructure® IPT boards.

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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pSessionInfo</b>	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 <a href="#">IPM_SESSION_INFO</a> for details.
<b>usMode</b>	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 when using an Intel NetStructure DM/IP board.
- **ipm\_GetSessionInfo()** is not supported on Intel NetStructure IPT boards. If called, it returns zeroes.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

**EIPM\_BADPARAM**  
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_getevtdata();

    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:** DM/IP, IPT

### ■ Description

The **ipm\_GetXmitSlot()** function returns TDM time slot information for an IP channel.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pTimeslotInfo</b>	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 <a href="#">SC_TSINFO</a> for details.
<b>usMode</b>	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\_BADPARAM**  
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_getevtttype();
    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\_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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pTimeslotInfo</b>	pointer to structure that describes the time slot number, time slot type, and bus encoding format See <a href="#">SC_TSINFO</a> for details.
<b>usMode</b>	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 Intel telecom libraries may not support asynchronous execution of the similar **xx\_Listen** and **xx\_UnListen** functions.
- On Intel NetStructure® DM/IP boards, in a configuration where a network interface device listens to the same TDM bus time slot device as a local, on-board voice device or other media device, the “sharing of time slot” (SOT) algorithm applies. This algorithm imposes limitations on the order and sequence of “listens” and “unlistens” between network and media devices. For details on application development rules and guidelines regarding the SOT algorithm, see the technical note at:  
<http://resource.intel.com/telecom/support/tnotes/tnbyos/2000/tn043.htm>

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

EIPM\_BADPARAM

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_enbhdr( 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;
    }
}

```

*ipm\_Listen( ) — connect an IP channel to a TDM time slot*



■ **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:** DM/IP

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pMediaInfo</b>	pointer to structure that contains local channel RTP/RTCP ports and IP address information (or T.38 port and IP address information) See the <a href="#">IPM_MEDIA_INFO</a> data structure page for details.

Parameter	Description
<b>eDirection</b>	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: <ul style="list-style-type: none"> <li>• DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data</li> <li>• DATA_IP_SENDOONLY – send data to the IP network but do not receive data</li> <li>• DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM</li> <li>• DATA_IP_INACTIVE – allow RTCP while blocking RTP or T.38 packets</li> <li>• DATA_MULTICAST_SERVER – multicast server mode</li> </ul>
<b>usMode</b>	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\_BADPARAM

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

## ■ 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, "111.21.0.9");

    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_FRAME_SIZE) 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_FRAME_SIZE) 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, change destination and direction
.
*/

MediaInfo.unCount = 2;
MediaInfo.MediaData[0].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTP_INFO;
MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.10");

MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
MediaInfo.MediaData[1].mediaInfo.PortInfo.unPortId = 2329;
strcpy(MediaInfo.MediaData[1].mediaInfo.PortInfo.cIPAddress, "111.21.0.10");

if (ipm_ModifyMedia(nDeviceHandle, &MediaInfo, DATA_IP_SENDOONLY, 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:** DM/IP, IPT

### ■ 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
<b>szDeviceName</b>	pointer to device name to open IP Media channel device: <b>ipmBxCy</b> where <b>x</b> is the unique logical board number and <b>y</b> is the media device channel number. Board device: <b>ipmBx</b> where <b>x</b> is the unique logical board number.
<b>pOpenInfo</b>	set to NULL; reserved for future use
<b>usMode</b>	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_enbhdr( 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\_Ping( )

**Name:** int ipm\_Ping(nDeviceHandle, \*pPingParameter, \*pPingInfo, usMode)

**Inputs:**

int nDeviceHandle	• board device handle
PIPM_PING_PARM *pPingParameter	• pointer to an array of ping parameter structures
IPM_PING_INFO *pPingInfo	• pointer to ping 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:** IPT

### ■ Description

The **ipm\_Ping( )** function generates a “ping” message to a remote IP address from an Ethernet\* interface. Typically used for testing and debugging, applications send a ping message and expect a response to be returned. The “ping” functionality operates on a per-board basis.

**Note:** This function is not supported on Intel NetStructure DM/IP Series boards.

Parameter	Description
<b>nDeviceHandle</b>	handle of the board device <b>ipmBx</b> , where <b>x</b> is the unique logical board number
<b>pPingParameter</b>	pointer to an array of ping parameter structures See <a href="#">IPM_PING_PARM</a> for details.
<b>pPingInfo</b>	pointer to structure that is filled with ping results upon successful return See <a href="#">IPM_PING_INFO</a> for details.
<b>usMode</b>	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

### ■ Termination Events

IPMEV\_PING

Indicates successful completion; that is, ping response information was returned. Use SRL functions to retrieve [IPM\\_PING\\_INFO](#) structure fields.

IPMEV\_ERROR

Indicates that the function failed.

## ■ Cautions

You must specify both a remote and a local IP address in the `IPM_PING_PARM` structure or this function will fail.

## ■ Errors

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

`EIPM_BADPARAM`  
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_enbhdr( EV_ANYDEV ,EV_ANYEVT , (HDLR) CheckEvent);

    /*
     *
     * Main Processing
     *
     */

    /*
     * ASSUMPTION: A valid nDeviceHandle was obtained from prior
     * call to ipm_Open() for a board device.
     */
    IPM_PING_PARM PingParameter;
    strcpy(PingParameter.cRemoteIPAddress, "192.168.1.16");
    strcpy(PingParameter.cLocalIPAddress, "192.168.1.16");
    PingParameter.ulNumOfPings = 0;
    PingParameter.ulPacketSize = 0;
    PingParameter.ulTimeout = 0;
    if(ipm_Ping(nDeviceHandle, &PingParameter, NULL, EV_ASYNC)==-1)
    {
        printf("ipm_Ping failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
         *
         * Perform Error Processing
         *
         */
    }
}
```

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

void CheckEvent()
{
    int nEventType = sr_getevtttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_PING_INFO* pPingInfo;

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

        /* Expected reply to ipm_Ping */
        case IPMEV_PING:
            pPingInfo = (IPM_PING_INFO*)pVoid;
            printf("Received IPMEV_PING for device = %s\n", ATDV_NAMEP(nDeviceID));
            printf("Packets sent=%u, Packets received=%u, Packets lost=%u\n",
                pPingInfo->unPacketsSent,
                pPingInfo->unPacketsReceived,
                pPingInfo->unPacketsLost);
            break;

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

#### ■ See Also

None.



## 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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pDigitInfo</b>	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 <b>unNumberOfDigits</b> field to indicate how many <b>IPMEV_DIGITS_RECEIVED</b> events the application must process.  See <a href="#">IPM_DIGIT_INFO</a> for details.  <b>Note:</b> The <b>cDigits[MAX_IP_DIGITS]</b> field in the data structure pointed to by <b>pDigitInfo</b> is not used by the <b>ipm_ReceiveDigits( )</b> function in either SYNC or ASYNC mode. The incoming digit is delivered asynchronously in the <a href="#">IPM_DIGIT_INFO</a> data structure associated with an <b>IPMEV_DIGITS_RECEIVED</b> event.
<b>usMode</b>	operation mode  Set to <b>EV_ASYNC</b> for asynchronous execution or to <b>EV_SYNC</b> 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 *IP Media Library API Programming Guide*.
- On Intel NetStructure® DM/IP boards, digits are only received when an RTP session is active; if two ipm devices are routed together, you must start an RTP session before digits can be sent and received. On Intel NetStructure® IPT boards, digits may be sent and received on ipm devices that are routed together regardless of whether an RTP session is active.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR( )** and **ATDV\_ERRMSGP( )** to return one of the following errors:

### EIPM\_BADPARAM

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_enbhdldr( 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:** srlib.h  
ipmlib.h

**Category:** QoS

**Mode:** asynchronous or synchronous

**Platform:** DM/IP

### ■ 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.

**Note:** This function is not supported on Intel NetStructure® IPT boards.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pQoSAlarmInfo</b>	pointer to <a href="#">IPM_QOS_ALARM_STATUS</a> structure which contains one or more <a href="#">IPM_QOS_ALARM_DATA</a> structures
<b>usMode</b>	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\_BADPARAM**

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_enbhdr( 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_getevtttype();
    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\_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:** DM/IP, IPT

### ■ Description

The **ipm\_SendDigits( )** function generates the supplied digits to the TDM bus.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pDigitInfo</b>	pointer to structure that contains digit type, direction, and digits; see <a href="#">IPM_DIGIT_INFO</a> 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 32 for Intel NetStructure® IPT boards and 16 for Intel NetStructure® DM/IP boards.
<b>usMode</b>	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.
- On Intel NetStructure® DM/IP boards, digits are only exchanged when an RTP session is active; if two ipm devices are routed together, you must start an RTP session before digits can be sent and received. On Intel NetStructure IPT boards, digits may be sent and received between ipm devices that are routed together regardless of whether an RTP session is active.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

**EIPM\_BADPARAM**

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_enbhdr( 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:** DM/IP, IPT

### ■ Description

The **ipm\_SetParm()** function sets values for the specified parameter.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP media device
<b>pParmInfo</b>	pointer to structure that contains IP channel parameter values See the <a href="#">IPM_PARM_INFO</a> data structure page for details.
<b>usMode</b>	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\_BADPARAM

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_enbhdr( 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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media channel device
<b>pQoSThresholdInfo</b>	pointer to <a href="#">IPM_QOS_THRESHOLD_INFO</a> structure which contains one or more <a href="#">IPM_QOS_THRESHOLD_DATA</a> structures with the threshold settings to be set.  Note that when an application needs to specify any given field in an <a href="#">IPM_QOS_THRESHOLD_DATA</a> structure, it must populate <i>all</i> fields in the structure even if those fields are to remain at their default values.
<b>usMode</b>	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\_BADPARAM**  
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_enbhdr( 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_getevtttype();
    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\_SetRemoteMediaInfo()

**Name:** int ipm\_SetRemoteMediaInfo(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:** DM/IP, IPT

### ■ Description

**Note:** The **ipm\_SetRemoteMediaInfo()** function is deprecated and is included in the library for backwards compatibility only. Application developers should use the **ipm\_StartMedia()** function instead of **ipm\_SetRemoteMediaInfo()**.

The **ipm\_SetRemoteMediaInfo()** function sets media properties and starts the session. This function allows the application to set the remote and local connectivity selections.

**ipm\_SetRemoteMediaInfo()** also starts RTP streaming. The remote RTP/ RTCP port information and coder information is provided in the IPM\_MEDIA\_INFO structure.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pMediaInfo</b>	media information data structure; see <a href="#">IPM_MEDIA_INFO</a> for details Applications can define the following: <ul style="list-style-type: none"> <li>• local transmit coder and remote transmit coder</li> <li>• local and remote RTP/RTCP ports</li> <li>• local and remote IP address</li> <li>• local and remote T.38 ports</li> </ul>

Parameter	Description
<b>eDirection</b>	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values that are valid for this function: <ul style="list-style-type: none"> <li>• DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data (supported for DM/IP boards only)</li> <li>• DATA_IP_SENDOONLY – send data to the IP network but do not receive data (supported for DM/IP boards only)</li> <li>• DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM</li> <li>• DATA_MULTICAST_SERVER – multicast server mode (supported for DM/IP boards only)</li> </ul>
<b>usMode</b>	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

## ■ Termination Events

### IPMEV\_SET\_REMOTE\_MEDIA\_INFO

Indicates successful completion; that is, media information was set and the session has been started. Use SRL functions to retrieve IPM\_MEDIA\_INFO structure fields.

### IPMEV\_ERROR

Indicates that the function failed.

## ■ Cautions

- The application must wait until this function completes before calling [ipm\\_Listen\(\)](#).
- See [IPM\\_CODER\\_INFO](#), on page 103 for limitations on coder type, frame size, and frames per packet settings.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR( )** and **ATDV\_ERRMSGP( )** to return one of the following errors:

### EIPM\_BADPARAM

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 <string>
#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
     *
     */

    /*
     * 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_FRAME_SIZE) 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_FRAME_SIZE) 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_SetRemoteMediaInfo(nDeviceHandle, &MediaInfo, DATA_IP_TDM_BIDIRECTIONAL,
        EV_ASYNC) == -1)
    {
        printf("ipm_SetRemoteMediaInfo 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_SetRemoteMediaInfo */
        case IPMEV_SET_REMOTE_MEDIA_INFO:
            printf("Received IPMEV_SET_REMOTE_MEDIA_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\\_GetLocalMediaInfo\(\)](#)
- [ipm\\_StartMedia\(\)](#)

## 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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pMediaInfo</b>	media information data structure; see <a href="#">IPM_MEDIA_INFO</a> for details Applications can define the following: <ul style="list-style-type: none"> <li>• local transmit coder and remote transmit coder</li> <li>• local and remote RTP/RTCP port</li> <li>• local and remote IP address</li> <li>• local and remote T.38 port</li> </ul>

Parameter	Description
<b>eDirection</b>	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: <ul style="list-style-type: none"> <li>• DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data (supported for DM/IP boards only)</li> <li>• DATA_IP_SENDOONLY – send data to the IP network but do not receive data (supported for DM/IP boards only)</li> <li>• DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM</li> <li>• DATA_MULTICAST_SERVER – multicast server mode (supported for DM/IP boards only)</li> </ul>
<b>usMode</b>	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. Use the SRL function to retrieve the IPM\_MEDIA\_INFO structure fields.

### 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.

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

### EIPM\_BADPARAM

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 <string>
#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
     *
     */

    /*
     * 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, "192.168.0.9");

    MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
    MediaInfo.MediaData[1].mediaInfo.PortInfo.unPortId = 2329;
    strcpy(MediaInfo.MediaData[1].mediaInfo.PortInfo.cIPAddress, "192.168.0.9");

    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_FRAME_SIZE) 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_FRAME_SIZE) 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_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:** DM/IP, IPT

### ■ 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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>eOperation</b>	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: <ul style="list-style-type: none"> <li>• STOP_RECEIVE_DIGITS – stop receiving digits</li> <li>• STOP_MEDIA – operation of media streaming session. This enumeration disconnects the session. The application must call <a href="#">ipm_StartMedia( )</a> to start a new session.</li> <li>• STOP_ALL – stop all operations</li> </ul>
<b>usMode</b>	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\_BADPARAM**  
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_enbhdr( 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_getevtttype();
    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:** DM/IP, IPT

### ■ Description

The **ipm\_UnListen( )** function stops listening to the TDM time slot specified in a previous call to **ipm\_Listen( )**. When **ipm\_Stop( )** is called to stop a media session on an Intel NetStructure<sup>®</sup> DM/IP board, **ipm\_UnListen( )** is called automatically.

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
<b>nDeviceHandle</b>	handle of the IP Media device
<b>usMode</b>	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 Intel telecom libraries may not support asynchronous execution of the similar xx\_Listen and xx\_UnListen functions.
- On Intel NetStructure DM/IP boards, in a configuration where a network interface device listens to the same TDM bus time slot device as a local, on-board voice device or other media

device, the “sharing of time slot” (SOT) algorithm applies. This algorithm imposes limitations on the order and sequence of “listens” and “unlistens” between network and media devices. For details on application development rules and guidelines regarding the SOT algorithm, see the technical note at:

<http://resource.intel.com/telecom/support/tnotes/tnbyos/2000/tn043.htm>

## ■ Errors

If the function returns -1 to indicate failure, call **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()** to return one of the following errors:

**EIPM\_BADPARAM**  
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_getevtttype();
    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\(\)](#)

This chapter describes the events that are returned by the IP Media software 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 IP Media software 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 IP Media library 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 IP Media 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 *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\_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\_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\_PING**

Termination event for [ipm\\_Ping\(\)](#). IPM\_PING\_INFO contains data. Indicates ping response has been returned.

**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 *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\_SECURITY\_KEYS\_LIFETIMEABOUTTOEXPIRE**

Occurs when the lifetime of the SRTP session key currently in use is about to expire, that is, there are only “X” number of packets left before the key expires.



**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.



This chapter alphabetically lists the data structures used by 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 .....	100
• IPM_CLOSE_INFO .....	102
• IPM_CODER_INFO .....	103
• IPM_DIGIT_INFO .....	107
• IPM_EVENT_INFO .....	108
• IPM_FAX_SIGNAL .....	109
• IPM_MEDIA .....	110
• IPM_MEDIA_INFO .....	111
• IPM_OPEN_INFO .....	112
• IPM_PARM_INFO .....	113
• IPM_PING_INFO .....	115
• IPM_PING_PARM .....	116
• IPM_PORT_INFO .....	117
• IPM_QOS_ALARM_DATA .....	118
• IPM_QOS_ALARM_STATUS .....	119
• IPM_QOS_SESSION_INFO .....	120
• IPM_QOS_THRESHOLD_DATA .....	121
• IPM_QOS_THRESHOLD_INFO .....	124
• IPM_RFC2833_SIGNALID_INFO .....	125
• IPM_RTCP_SESSION_INFO .....	126
• IPM_SESSION_INFO .....	128
• SC_TSINFO .....	129

## 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 IP Media library. Note that this same data structure definition is used in other Intel telephony 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\_DFDM3 – Intel NetStructure DM/IP board
- CT\_NETSTRUCTIP – Intel NetStructure IPT board

**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\_H100 – H.100 bus
- CT\_H110 – H.110 bus

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\_protype

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\_CODER\_INFO

```
typedef struct ipm_coder_info_tag
{
    eIPM_CODER_TYPE          eCoderType;          /* The coder Type */
    eIPM_CODER_FRAME_SIZE    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_CODER_INFO, *PIPM_CODER_INFO;
```

### Description

This structure contains the coder properties that will be used in an IP session. IPM\_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\(\)](#), [ipm\\_ModifyMedia\(\)](#), and [ipm\\_StartMedia\(\)](#) functions.

Appropriate values for IPM\_CODER\_INFO fields depend on the board that is being used. Table 2 and Table 3 list supported coders and coder properties for Intel NetStructure® IPT Series boards and Intel NetStructure® DM/IP Series boards, respectively.

### Field Descriptions

The fields of the IPM\_CODER\_INFO data structure are described as follows. Refer to Table 2 and Table 3 for product-specific and coder-specific guidelines for filling in these fields.

#### eCoderType

type of coder to be used for streaming media operations Platform-specific values for this field are listed in Table 2 and Table 3.

The following values are defined:

- CODER\_TYPE\_AMRNB\_4\_75k – GSM AMR-NB, 4.75 kbps
- CODER\_TYPE\_AMRNB\_5\_15k – GSM AMR-NB, 5.15 kbps
- CODER\_TYPE\_AMRNB\_5\_9k – GSM AMR-NB, 5.9 kbps
- CODER\_TYPE\_AMRNB\_6\_7k – GSM AMR-NB, 6.7 kbps
- CODER\_TYPE\_AMRNB\_7\_4k – GSM AMR-NB, 7.4 kbps
- CODER\_TYPE\_AMRNB\_7\_95k – GSM AMR-NB, 7.95 kbps
- CODER\_TYPE\_AMRNB\_10\_2k – GSM AMR-NB, 10.2 kbps
- CODER\_TYPE\_AMRNB\_12\_2k – GSM AMR-NB, 12.2 kbps
- CODER\_TYPE\_G711ALAW64K – G.711, A-law, 64 kbps
- CODER\_TYPE\_G711ULAW64K – G.711, mu-law, 64 kbps
- CODER\_TYPE\_G7231\_5\_3K – G.723.1, 5.3 kbps
- CODER\_TYPE\_G726\_32K – G.726.3, 32 kbps
- CODER\_TYPE\_G729ANNEXA – G.729 Annex A
- CODER\_TYPE\_G729ANNEXAWANNEXB – G.729 Annex A with Annex B
- CODER\_TYPE\_GSMFULLRATE – GSM (TIPHON\*), full rate

#### eFrameSize

size of frame for coders that support multiple frame sizes—currently G.711 and G.726 coders only. (All 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 **eIPM\_CODER\_FRAMESIZE** to specify that value.

The **eIPM\_CODER\_FRAMESIZE** data type is an enumeration which specifies the frame size for G.711 and G.726 coders only. The following values for **eIPM\_CODER\_FRAMESIZE** are enumerated, and the platform-specific supported values are listed in Table 2 and Table 3:

- **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 for this field are listed in Table 2 and Table 3. 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:

- **CODER\_VAD\_DISABLE** – VAD is OFF
- **CODER\_VAD\_ENABLE** – VAD is ON

#### **unCoderPayloadType**

RTP header payload type using RFC 3551 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.

**Note:** Coders that use payload types in the dynamic range (96 to 127) must be negotiated by the application and must agree.

#### **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 Coder Properties for Intel NetStructure IPT Series Boards**

<b>eCoderType</b>	<b>Frame Size (ms)</b>	<b>Frames per Packet (fpp)</b>	<b>eVadEnable Value</b>
CODER_TYPE_AMRNB_4_75k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_5_15k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_5_9k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
<b>NOTES:</b> 1. IPT Series boards only support symmetric coders; that is, transmit and receive coder specifications must be identical. 2. Applications must explicitly enable VAD even though G.723a+b implicitly supports VAD.			



**Table 2. Supported Coder Properties for Intel NetStructure IPT Series Boards (Continued)**

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_AMRNB_6_7k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_7_4k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_7_95k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_10_2k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_AMRNB_12_2k	(fixed at 20)	1, 2, or 3	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ALAW64K	10, 20, or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ULAW64K	10, 20, or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G7231_5_3K	(fixed at 30)	1, 2, 3, or 4	Either value
CODER_TYPE_G7231_6_3K	(fixed at 30)	1, 2, 3, 4	Either value
CODER_TYPE_G726_32K	10	1, 2, or 3	Must be CODER_VAD_DISABLE
	20	1 or 2	
	30	1	
CODER_TYPE_G729ANNEXA	(fixed at 10)	1, 2, 3, or 4	Must be CODER_VAD_DISABLE
CODER_TYPE_G729ANNEXAWANNEXB	(fixed at 10)	1, 2, 3, or 4	Must be CODER_VAD_ENABLE <sup>2</sup>
<b>NOTES:</b> 1. IPT Series boards only support symmetric coders; that is, transmit and receive coder specifications must be identical. 2. Applications must explicitly enable VAD even though G.723a+b implicitly supports VAD.			

Table 3. Supported Coder Properties for Intel NetStructure DM/IP Series Boards

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_G711ALAW64K	20 or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ULAW64K	20 or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G7231_5_3K	(fixed at 30)	1, 2, or 3	Either value
CODER_TYPE_G7231_6_3K	(fixed at 30)	1, 2, or 3	Either value
CODER_TYPE_G729ANNEXA	(fixed at 10)	2, 3, or 4	Must be CODER_VAD_DISABLE
CODER_TYPE_G729ANNEXAWANNEXB	(fixed at 10)	2, 3, or 4	Must be CODER_VAD_ENABLED <sup>2</sup>
CODER_TYPE_GSMFULLRATE <sup>3</sup>	(fixed at 20)	1, 2, or 3	Either value
<b>NOTES:</b> 1. DM/IP Series boards only support symmetric coders; that is, transmit and receive coder specifications must be identical. 2. Applications must explicitly enable VAD even though G.723a+b implicitly supports VAD. 3. GSM Telecommunications and Internet Protocol Harmonization over Networks (TIPHON*) is a sub-group of the European Telecommunications Standards Institute (ETSI) GSM specification.			

## 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 for Intel NetStructure® DM/IP boards or 32 for Intel NetStructure® IPT boards. 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() 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\_MEDIA

```

struct IPM_MEDIA_tag
{
    eIPM_MEDIA_TYPE eMediaType;
    union
    {
        IPM_PORT_INFO      PortInfo;          /* RTP port information */
        IPM_CODER_INFO      CoderInfo;         /* Audio coder information */
        IPM_FAX_SIGNAL      FaxSignal;         /* Fax signal information */
    }
} IPM_MEDIA, *PIPM_MEDIA;

```

### Description

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

- [IPM\\_PORT\\_INFO](#)
- [IPM\\_CODER\\_INFO](#)
- [IPM\\_FAX\\_SIGNAL](#)

This structure is a child of the [IPM\\_MEDIA\\_INFO](#) structure which is used by the [ipm\\_GetLocalMediaInfo\(\)](#), [ipm\\_ModifyMedia\(\)](#) and [ipm\\_StartMedia\(\)](#) functions.

### 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:

- MEDIATYPE\_FAX\_SIGNAL\_INFO – fax signal information to be transmitted towards IP during fax transmissions
- MEDIATYPE\_AUDIO\_LOCAL\_CODER\_INFO – local receive coder information
- MEDIATYPE\_AUDIO\_LOCAL\_RTCP\_INFO – local RTCP port information
- MEDIATYPE\_AUDIO\_LOCAL\_RTP\_INFO – local RTP port information
- MEDIATYPE\_LOCAL\_UDPTL\_T38\_INFO – local UDP packet T.38 information
- MEDIATYPE\_AUDIO\_REMOTE\_CODER\_INFO – remote receive coder information
- MEDIATYPE\_AUDIO\_REMOTE\_RTCP\_INFO – remote RTCP port information
- MEDIATYPE\_AUDIO\_REMOTE\_RTP\_INFO – remote RTP port information
- MEDIATYPE\_REMOTE\_UDPTL\_T38\_INFO – remote UDP packet T.38 information

**PortInfo**

reference to RTP port information data structure, type [IPM\\_PORT\\_INFO](#)

**CoderInfo**

reference to audio coder information data structure, type [IPM\\_CODER\\_INFO](#)

**FaxSignal**

reference to fax signal data structure, type [IPM\\_FAX\\_SIGNAL](#)

## 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\_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 4 for supported types and corresponding values.

**pvParmValue**

pointer to the value of the parameter. See Table 4 for supported values for each parameter type.

**Table 4. eIPM\_PARM Parameters and Values**

eIP_PARM Define	Description and Values
PARMCH_AGCACTIVE	Enables/disables automatic gain for DM/IP boards only. Type: eIPM_AGCACTIVE (enumeration). Values: <ul style="list-style-type: none"> <li>AGCACTIVE_OFF (default)</li> <li>AGCACTIVE_ON</li> </ul>
PARMCH_DTMFXFERMODE	Specifies DTMF transfer mode Type: eIPM_DTMFXFERMODE (enumeration). Values: <ul style="list-style-type: none"> <li>DTMFXFERMODE_INBAND – in-band (default)</li> <li>DTMFXFERMODE_OUTOFBAND – out-of-band</li> <li>DTMFXFERMODE_RFC2833 – RFC2833</li> </ul> <b>Note:</b> 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 for DM/IP and IPT boards. Type: eIPM_ACTIVE (enumeration). Values: <ul style="list-style-type: none"> <li>ECACTIVE_OFF</li> <li>ECACTIVE_ON (default)</li> </ul>

Table 4. eIPM\_PARM Parameters and Values (Continued)

eIP_PARM Define	Description and Values
PARMCH_ECHOTAIL	<p>Specifies echo tail length value for DM/IP and IPT boards. Type: eIPM_ECHO_TAIL (enumeration)</p> <ul style="list-style-type: none"> <li>•</li> </ul> <p>Supported values for Intel NetStructure DM/IP boards:</p> <ul style="list-style-type: none"> <li>• ECHO_TAIL_NONE</li> <li>• ECHO_TAIL_8</li> <li>• ECHO_TAIL_16</li> <li>• ECHO_TAIL_32</li> <li>• ECHO_TAIL_48 (default)</li> </ul> <p>Supported values for Intel NetStructure IPT boards:</p> <ul style="list-style-type: none"> <li>• ECHO_TAIL_NONE (default)</li> <li>• ECHO_TAIL_8</li> <li>• ECHO_TAIL_16</li> <li>• ECHO_TAIL_32</li> <li>• ECHO_TAIL_48</li> <li>• ECHO_TAIL_64</li> <li>• ECHO_TAIL_96</li> <li>• ECHO_TAIL_128</li> </ul>
PARMCH_RFC2833EVT_RX_PLT	<p>Specifies RFC2833 event receive payload type. Type: unsigned char. Valid values: 96 to 127. Default: 101.</p>
PARMCH_RFC2833EVT_TX_PLT	<p>Specifies RFC2833 event transmit payload type. Type: unsigned char. Valid values: 96 to 127. Default: 101.</p>
PARMCH_RFC2833GEN_TO_IP	<p>Enables/disables sending RFC2833 to IP Type: eIPM_RFC2833GEN_TO_IP. Values:</p> <ul style="list-style-type: none"> <li>• RFC2833GEN_TO_IP_OFF</li> <li>• RFC2833GEN_TO_IP_ON (default)</li> </ul>
PARMCH_RFC2833GEN_TO_TDM	<p>Enables/disables converting RFC2833 to TDM signal. Type: eIPM_RFC2833GEN_TO_TDM. Values:</p> <ul style="list-style-type: none"> <li>• RFC2833GEN_TO_TDM_OFF</li> <li>• RFC2833GEN_TO_TDM_ON (default)</li> </ul>
PARMCH_RFC2833REDLEVEL	<p>Specifies RFC2833 redundancy level for DM/IP boards only. Type: eIPM_RFC2833REDLEVEL. Values:</p> <ul style="list-style-type: none"> <li>• RFC2833REDLEVEL_0</li> <li>• RFC2833REDLEVEL_1</li> <li>• RFC2833REDLEVEL_2</li> <li>• RFC2833REDLEVEL_3</li> </ul>
PARMCH_TOS	<p>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.</p>
PARMCH_TTL	<p>Specifies time-to-live for multicast. Type: char. Valid values: 0 to 255. Default: 1. <b>Note:</b> Not supported on Intel NetStructure IPT boards.</p>

## IPM\_PING\_INFO

```
typedef struct ipm_ping_info_tag
{
    unsigned int unPacketsSent;
    unsigned int unPacketsReceived;
    unsigned int unPacketsLost;

    float fRoundTripMin;      /* Time values in mSec */
    float fRoundTripAvg;
    float fRoundTripMax;
}IPM_PING_INFO, * PIPM_PING_INFO ;
```

### ■ Description

This structure contains ping response information. The structure is used by the [ipm\\_Ping\(\)](#) function, which is only supported for Intel NetStructure® IPT Series boards.

### ■ Field Descriptions

The fields of the IPM\_PING\_INFO data structure are described as follows:

unPacketsSent	number of packets sent
unPacketsReceived	number of packets received
unPacketsLost	number of packets lost
fRoundTripMin	minimum round trip time in msec
fRoundTripAvg	average round trip time in msec
fRoundTripMax	maximum round trip time in msec

## IPM\_PING\_PARM

```
typedef struct ipm_ping_parameter_tag
{
    char  cRemoteIPAddress[IP_ADDR_SIZE]; /* Destination IP Address */
    char  cLocalIPAddress[IP_ADDR_SIZE];  /* Local IP Address */
    unsigned long ulNumOfPings;           /* RFU - Number of Echo Requests to send */
    unsigned long ulPacketSize;           /* RFU - Number of data bytes to be sent */
    unsigned long ulTimeout;              /* RFU - mSec Timeout to wait for each reply */
} IPM_PING_PARM, * PIPM_PING_PARM;
```

### Description

This structure contains ping parameter information. The structure is used by the [ipm\\_Ping\(\)](#) function, which is only supported for Intel NetStructure® IPT Series boards.

**Note:** For a board device, the value for cLocalIPAddress can be obtained by calling [ipm\\_GetParm\(\)](#). For a channel device, [ipm\\_GetLocalMediaInfo\(\)](#) should be used. However, the IP addresses returned from [ipm\\_GetParm\(\)](#) will work for channel devices.

### Field Descriptions

The fields of the IPM\_PING\_PARM data structure are described as follows:

cRemoteIPAddress[IP\_ADDR\_SIZE]

destination IP address; null-terminated string formatted as standard dotted-decimal IP address

cLocalIPAddress[IP\_ADDR\_SIZE]

local board IP address; null-terminated string formatted as standard dotted-decimal IP address

ulNumOfPings

reserved for future use (RFU); set to 0

ulPacketSize

reserved for future use (RFU); set to 0

ulTimeout

reserved for future use (RFU); set to 0

## 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.

For Intel NetStructure® DM/IP boards, the library generates a IPMEV\_QOS\_ALARM alarm event with ALARM\_STATE\_ON when a QoS fault threshold is exceeded, and it generates a QoS alarm event with ALARM\_STATE\_OFF when the fault measurement returns to a sub-threshold level.

For Intel NetStructure® IPT boards, the library generates a IPMEV\_QOS\_ALARM alarm event with ALARM\_STATE\_ON when a channel-level QoS fault threshold is exceeded but does not generate an event when the fault condition returns to programmed levels. For the board-level QOSTYPE\_NETWORKFAILURE alarm, a single alarm event with ALARM\_STATE\_ON is generated within 1 second when the board's connection to the RTP network is disrupted, and a single alarm event with ALARM\_STATE\_OFF is generated within 1 second when the network connection is restored.

### 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 – QoS alarm for excessive lost packets (IPT boards only)
- QOSTYPE\_NETWORKFAILURE – board-level alarm for RTP network disruption (IPT boards only)
- QOSTYPE\_ROUNDTRIPLATENCY – QoS alarm for RTP packet latency (IPT boards only)

#### 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 – percent of lost packets since beginning of call (IPT boards only)
- QOSTYPE\_ROUNDTRIPLATENCY – RTP packet latency (IPT boards only)

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 5 and Table 6 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\_THRESHOLD 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 (IPT boards only)
- QOSTYPE\_ROUNDTRIPLATENCY – RTP packet latency (IPT boards only)

#### 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.

**Note:** This field is not supported on IPT boards and must be set to 0.

#### 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 supported on IPT boards 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 supported on IPT boards 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) for DM/IP boards, 0 to 255 for IPT boards
- QOSTYPE\_LOSTPACKET – allowable percentage of lost packets for IPT boards only. Range: 0 to 100 (%)
- QOSTYPE\_ROUNDTRIPLATENCY – allowable RPT packet latency interval, in ms, for IPT boards only. Range: 0 to 255

**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 field is not supported on IPT boards 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 field is not supported on IPT boards and must be set to 0.

**Table 5. Quality of Service Parameter Defaults for IPT Boards**

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold <sup>1</sup>	% Success Threshold	% Fail Threshold
Jitter	0	0	0	30 (ms)	0	0
Lost Packets	0	0	0	100 (%)	0	0
Round-trip Latency	0	0	0	950 (ms)	0	0
Notes: 1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.						

**Table 6. Quality of Service Parameter Defaults for DM/IP Boards**

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold <sup>1</sup>	% Success Threshold	% Fail Threshold
Jitter	5000	20000	60000	60 (ms)	40	40
Lost Packets	n/a	n/a1	n/a	n/a	n/a	n/a
Round-trip Latency	n/a	n/a	n/a	n/a	n/a	n/a
Notes: 1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.						



QoS debouncing for DM/IP boards 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:

$$\text{count} = \text{int}(\text{int}(\text{unDebounceOn}/\text{unTimeInterval}) * (\text{unPercentFailThreshold}/100))$$

For QoS alarm-off debouncing:

$$\text{count} = \text{int}(\text{int}(\text{unDebounceOff}/\text{unTimeInterval}) * (\text{unPercentSuccessThreshold}/100))$$

## 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 7.

**Table 7. 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
SIGNAL_ID_EVENT_LINE_RINGING_TONE	0x46

**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.

**Note:** This structure is not fully supported for Intel NetStructure IPT boards. Only the unLocalRR\_CumulativeLost, unLocalSR\_TxOctets and unRemoteSR\_TxOctets fields contain valid values. All other fields are not valid and contain the value 0.

### 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\_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.

**Note:** This structure is not supported for Intel NetStructure IPT boards. See the [IPM\\_RTCP\\_SESSION\\_INFO](#) structure for details.

### ■ 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



## 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



This chapter describes the error/cause codes supported by the IP Media software error library, *ipmerror.h*. All 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 *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\_BADPARAM

Bad argument or parameter. All IP Media library functions except **ipm\_Open()**.

## EIPM\_BUSY

Device busy. **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\_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\\_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\\_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\\_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  $\mu$ sec) 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





## C

coder support  
     DM/IP Series boards 106  
     HMP software 125  
     IPT Series boards 104  
 coder types 103  
 conventions, device names 59  
 CT\_DEVINFO data structure 100

## D

data structures  
     CT\_DEVINFO 100  
     IPM\_CLOSE\_INFO 102  
     IPM\_CODER\_INFO 103  
     IPM\_DIGIT\_INFO 107  
     IPM\_EVENT\_INFO 108  
     IPM\_MEDIA 110  
     IPM\_MEDIA\_INFO 111  
     IPM\_OPEN\_INFO 112  
     IPM\_PORT\_INFO 117  
     IPM\_QOS\_ALARM\_DATA 118  
     IPM\_QOS\_ALARM\_STATUS 119  
     IPM\_QOS\_SESSION\_INFO 120  
     IPM\_QOS\_THRESHOLD\_DATA 121  
     IPM\_QOS\_THRESHOLD\_INFO 124  
     IPM\_RTCP\_SESSION\_INFO 126  
     IPM\_SESSION\_INFO 128  
     IPM\_TIMESLOT\_INFO 129  
     SC\_TSINFO 129  
 Differentiated Services 114  
 DSCP field 114  
 DTMF transfer mode 113

## F

full duplex  
     parameter for 56, 86

## I

I/O Functions 14  
     ipm\_ReceiveDigits() 65  
     ipm\_SendDigits() 72  
 ipm\_Close() 18

IPM\_CLOSE\_INFO data structure 102  
 IPM\_CODER\_INFO data structure 103  
 IPM\_DIGIT\_INFO data structure 107  
 ipm\_DisableEvents() 20  
 ipm\_EnableEvents() 24  
 IPM\_EVENT\_INFO data structure 108  
 IPM\_FAX\_SIGNAL data structure 109  
 ipm\_GetCTInfo() 28  
 ipm\_GetLocalMediaInfo() 30  
 ipm\_GetParm() 34  
 ipm\_GetQoSAlarmStatus() 37  
 ipm\_GetQoSThreshold() 40  
 ipm\_GetSessionInfo() 44  
 ipm\_GetXmitSlot() 48  
 ipm\_Listen() 51  
 IPM\_MEDIA data structure 110  
 IPM\_MEDIA\_INFO data structure 111  
 ipm\_ModifyMedia() 55  
 ipm\_Open() 59  
 IPM\_OPEN\_INFO data structure 112  
 IPM\_PORT\_INFO data structure 117  
 IPM\_QOS\_ALARM\_DATA data structure 118  
 IPM\_QOS\_ALARM\_STATUS data structure 119  
 IPM\_QOS\_SESSION\_INFO data structure 120  
 IPM\_QOS\_THRESHOLD\_DATA data structure 121  
 IPM\_QOS\_THRESHOLD\_INFO data structure 124  
 ipm\_ReceiveDigits() 65  
 ipm\_ResetQoSAlarmStatus() 69  
 IPM\_RTCP\_SESSION\_INFO data structure 126  
 ipm\_SendDigits() 72  
 IPM\_SESSION\_INFO data structure 128  
 ipm\_SetParm() 75  
 ipm\_SetQoSThreshold() 78  
 ipm\_SetRemoteMediaInfo() 81  
 ipm\_StartMedia() 85  
 ipm\_Stop() 89  
 IPM\_TIMESLOT\_INFO data structure 129  
 ipm\_UnListen() 92  
 IPMEV\_DIGITS\_RECEIVED event 66  
 IPMEV\_MODIFY\_MEDIA 56  
 IPMEV\_MODIFY\_MEDIA\_FAIL 56

## M

Media Session Functions 14

- ipm\_GetCTInfo( ) 28
- ipm\_GetLocalMediaInfo( ) 30
- ipm\_GetSessionInfo( ) 44
- ipm\_ModifyMedia( ) 55
- ipm\_StartMedia( ) 85
- ipm\_Stop( ) 89

multicast server mode 56, 86

## N

naming convention, devices 59

## P

parameters for 113

PARMCH\_TOS 114

## Q

QoS Functions 14

- ipm\_GetQoSAlarmStatus( ) 37
- ipm\_GetQoSThreshold( ) 40
- ipm\_ResetQoSAlarmStatus( ) 69
- ipm\_SetQoSThreshold( ) 78

## R

RFC2833

- payload type parameters 114

## S

SC\_TSINFO data structure 129

System Control Functions 13

- ipm\_Close( ) 18
- ipm\_DisableEvents( ) 20
- ipm\_EnableEvents( ) 24
- ipm\_GetParm( ) 34
- ipm\_GetXmitSlot( ) 48
- ipm\_Listen( ) 51
- ipm\_Open( ) 59
- ipm\_Ping( ) 13
- ipm\_SetParm( ) 75
- ipm\_UnListen( ) 92

## T

TOS field 114

type 114