



# IP Media Library API for Host Media Processing

Library Reference

---

*September 2004*



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, or life sustaining applications.

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

This IP Media Library API for Host Media Processing 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 express written consent of Intel Corporation.

Copyright © 2003, Intel Corporation

BunnyPeople, Celeron, Chips, Dialogic, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel Centrino logo, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, Sound Mark, The Computer Inside., The Journey Inside, VTune, and Xircom 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: September 2004

Document Number: 05-2257-003

Intel Converged Communications, Inc.  
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 Products website at:  
<http://www.intel.com/design/network/products/telecom>

For **Sales Offices** and other contact information, visit the Where to Buy Intel Telecom Products page at:  
<http://www.intel.com/buy/wtb/wtb1028.htm>

# Contents

---

	<b>Revision History</b> .....	7
	<b>About This Publication</b> .....	9
<b>1</b>	<b>Function Summary by Category</b> .....	11
1.1	System Control Functions .....	11
1.2	I/O (Input/Output) Functions .....	12
1.3	Media Session Functions .....	12
1.4	Quality of Service (QoS) Functions .....	12
<b>2</b>	<b>Function Information</b> .....	13
2.1	Function Syntax Conventions .....	13
	ipm_Close( ) – close an IP channel device .....	14
	ipm_DisableEvents( ) – disable IP notification events .....	16
	ipm_EnableEvents( ) – enable IP notification events .....	20
	ipm_GetCTInfo( ) – return information about a voice channel of an IPM device .....	24
	ipm_GetLocalMediaInfo( ) – retrieve properties for the local media channel .....	26
	ipm_GetParm( ) – retrieve the current value of a parameter .....	29
	ipm_GetQoSAlarmStatus( ) – retrieve ON/OFF state of all QoS alarms .....	32
	ipm_GetQoSThreshold( ) – retrieve QoS alarm threshold settings .....	35
	ipm_GetSessionInfo( ) – retrieve statistics for a session .....	39
	ipm_GetXmitSlot( ) – return TDM time slot information for an IP channel .....	43
	ipm_Listen( ) – connect an IP channel to a TDM time slot .....	46
	ipm_Open( ) – open an IP channel device .....	49
	ipm_ReceiveDigits( ) – enable the IP channel to receive digits .....	52
	ipm_ResetQoSAlarmStatus( ) – reset QoS alarm(s) to the OFF state .....	56
	ipm_SendDigits( ) – generate supplied digits in the specified direction .....	59
	ipm_SendRFC2833SignalIDToIP( ) – send the supplied RFC 2833 signal .....	62
	ipm_SetParm( ) – set value for specified parameter .....	65
	ipm_SetQoSThreshold( ) – change QoS alarm threshold settings .....	68
	ipm_SetRemoteMediaInfo( ) – set media properties and starts the session .....	71
	ipm_StartMedia( ) – set media properties and start the session .....	75
	ipm_Stop( ) – stop operations on the specified IP channel .....	79
	ipm_UnListen( ) – stop listening to the TDM time slot .....	82
<b>3</b>	<b>Events</b> .....	85
<b>4</b>	<b>Data Structures</b> .....	89
	CT_DEVINFO – channel/time slot device information .....	90
	IPM_CLOSE_INFO – reserved for future use .....	92
	IPM_CODER_INFO – coder properties used in an IP session .....	93
	IPM_DIGIT_INFO – used to transfer digits over IP network and TDM bus .....	95
	IPM_EVENT_INFO – used for IP event notification .....	96
	IPM_FAX_SIGNAL – detected tone information definition .....	97

IPM_MEDIA – parent of port and coder info structures. . . . .	98
IPM_MEDIA_INFO – parent of IP_MEDIA, contains session info. . . . .	99
IPM_OPEN_INFO – reserved for future use. . . . .	100
IPM_PARM_INFO – used to set or retrieve parameters for an IP channel. . . . .	101
IPM_PORT_INFO – RTP and RTCP port properties . . . . .	102
IPM_QOS_ALARM_DATA – data associated with QoS alarms . . . . .	103
IPM_QOS_ALARM_STATUS – parent of QoS alarm data, contains alarm status. . . . .	104
IPM_QOS_SESSION_INFO – QoS statistics for an IP session . . . . .	105
IPM_QOS_THRESHOLD_DATA – QoS alarm threshold settings for an IP channel . . . . .	106
IPM_QOS_THRESHOLD_INFO – parent of threshold data structures. . . . .	108
IPM_RFC2833_SIGNALID_INFO – RFC 2833 signal ID and state info . . . . .	109
IPM_RTCP_SESSION_INFO – session information for RTCP. . . . .	111
IPM_SESSION_INFO – parent structure containing RTCP and QoS info . . . . .	113
SC_TSINFO – TDM bus (CT Bus) time slot information. . . . .	114
<b>5 Error Codes . . . . .</b>	<b>115</b>
<b>Glossary . . . . .</b>	<b>117</b>
<b>Index . . . . .</b>	<b>121</b>

## Tables

---

1	Supported Coder Properties for Host Media Processing .....	94
2	eIPM_PARM Parameters and Values .....	101
3	Quality of Service Parameter Defaults for Host Media Processing .....	107
4	eIPM_RFC2833_SIGNAL_ID Values .....	109



## Revision History

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

Document No.	Publication Date	Description of Revisions
05-2257-003	September 2004	<p><a href="#">ipm_GetQoSAlarmStatus( )</a> function (pg. 32): Corrected code example</p> <p><a href="#">ipm_GetSessionInfo( )</a> function (pg. 39): Added info on NULL pointer in asynch mode</p> <p><a href="#">ipm_GetXmitSlot( )</a> function (pg. 43): Added info on NULL pointer in asynch mode</p> <p><a href="#">ipm_Open( )</a> function (pg. 49): Removed caution that function must be called in synchronous mode</p> <p><a href="#">ipm_ReceiveDigits( )</a> function (pg. 52): Clarified usage of data structure. Added caution regarding active RTP session requirement for receiving digits.</p> <p><a href="#">ipm_SendDigits( )</a> function (pg. 59): Added info about maximum number of digits. Added caution regarding active RTP session requirement for sending digits.</p> <p><a href="#">ipm_StartMedia( )</a> function (pg. 75): Added caution to avoid setting IP address 0.0.0.0 (PTR#32986). Corrected name of completion event.</p> <p><a href="#">CT_DEVINFO</a> data structure (pg. 90): Removed much information irrelevant to the structure's use with IPML on HMP</p> <p><a href="#">IPM_DIGIT_INFO</a> data structure (pg. 95): Added info about maximum number of digits and send vs. receive usage differences</p> <p><a href="#">IPM_PARM_INFO</a> data structure (pg. 101): Removed descriptions of three unsupported RFC2833 parameters</p> <p><a href="#">IPM_PORT_INFO</a> data structure (pg. 102): Added caution to avoid setting IP address 0.0.0.0 (PTR#32986)</p>
05-2257-002	April 2004	<p>Made document HMP-specific by removing hardware-specific information, including "IP Media Function Support by Platform" section.</p> <p><a href="#">ipm_DisableEvents( )</a> function: Removed two unsupported events. Added <a href="#">EVT_RTCPTIMEOUT</a> and <a href="#">EVT_RTPTIMEOUT</a> QoS alarm events.</p> <p><a href="#">ipm_EnableEvents( )</a> function: Removed two unsupported events. Added <a href="#">EVT_RTCPTIMEOUT</a> and <a href="#">EVT_RTPTIMEOUT</a> QoS alarm events.</p> <p><a href="#">ipm_GetLocalMediaInfo( )</a> function: Corrected <a href="#">MEDIATYPE...</a> types referred to in the Cautions and Code Example sections</p> <p><a href="#">ipm_Ping( )</a>: Removed as unsupported</p> <p><a href="#">ipm_SendDigits( )</a> function: Documented as supported for HMP</p> <p><a href="#">ipm_SetQoSThreshold( )</a> function: Revised code example to use correct minimum value (100) for <a href="#">unTimeInterval</a></p> <p><a href="#">ipm_SetRemoteMediaInfo( )</a> function: Documented function as deprecated. Corrected code example.</p> <p><a href="#">ipm_StartMedia( )</a> function: Corrected code example</p> <p><a href="#">ipm_Stop( )</a> function: Removed two unsupported <a href="#">eIPM_STOP_OPERATION</a> values</p> <p><a href="#">IPM_CODER_INFO</a> data structure: Removed hardware-specific enum values and tables of supported coders. Updated table of supported coders.</p> <p><a href="#">IPM_PARM_INFO</a> data structure: Added <a href="#">PARMCH_RX_ADJVOLUME</a> and <a href="#">PARMCH_RX_ADJVOLUME</a> parameters. Expanded and corrected descriptions of all parameters. Removed hardware-specific enum values.</p>

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



## 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 instructions for developing applications using the IP Media Library.

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

### 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*
- <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 . . . . .](#) 11
- [I/O \(Input/Output\) Functions . . . . .](#) 12
- [Media Session Functions . . . . .](#) 12
- [Quality of Service \(QoS\) Functions . . . . .](#) 12

## 1.1 System Control Functions

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

### [\*\*ipm\\_Close\(\)\*\*](#)

closes an IP channel

### [\*\*ipm\\_DisableEvents\(\)\*\*](#)

disables IP notification events

### [\*\*ipm\\_EnableEvents\(\)\*\*](#)

enables IP notification events

### [\*\*ipm\\_GetParm\(\)\*\*](#)

returns IP channel parameters

### [\*\*ipm\\_GetXmitSlot\(\)\*\*](#)

returns TDM time slot information for an IP channel

### [\*\*ipm\\_Listen\(\)\*\*](#)

connects an IP channel to a TDM time slot

### [\*\*ipm\\_Open\(\)\*\*](#)

opens an IP channel and returns a handle

### [\*\*ipm\\_SetParm\(\)\*\*](#)

sets IP channel parameters

### [\*\*ipm\\_UnListen\(\)\*\*](#)

disconnects an IP channel from a TDM time slot

## 1.2 I/O (Input/Output) Functions

The following functions are used to transfer digits and data:

### **ipm\_ReceiveDigits()**

enables the IP channel to receive digits from the specified direction

### **ipm\_SendDigits()**

generates supplied digits in the specified direction

### **ipm\_SendRFC2833SignalIDToIP()**

sends the supplied RFC 2833 signal

## 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\_SetRemoteMediaInfo()**

This function is deprecated and included only for backwards compatibility; use

**ipm\_StartMedia()** instead.

### **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

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:** srllib.h  
ipmlib.h

**Category:** System Control

**Mode:** synchronous only

### ■ 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	• specifies 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

### ■ Description

The **ipm\_DisableEvents( )** function disables IP notification events. Some events are used for Quality of Service (QoS) notifications. Other events are used to indicate status, for example, if fax tone has been detected.

Notification events are different from asynchronous function termination events, such as IPMEV\_OPEN, which cannot be disabled. Once events are successfully disabled, if any events occur, the application is not notified.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pEvents</b>	<p>pointer to enumeration 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 – QoS alarm for excessive percentage of lost packets</li> <li>• EVT_JITTER – QoS alarm for excessive average jitter</li> <li>• EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity</li> <li>• EVT_RTPTIMEOUT – QoS alarm for RTP inactivity</li> <li>• EVT_FAXTONE – fax tone from TDM</li> <li>• EVT_RFC2833 – RFC 2833 event</li> <li>• EVT_T38CALLSTATE – T.38 call state event</li> </ul>

Parameter	Description
<b>unNumOfEvents</b>	number of events to disable
<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

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\_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	• specifies 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:** srlib.h  
ipmlib.h

**Category:** System Control

**Mode:** asynchronous or synchronous

### ■ Description

The **ipm\_EnableEvents( )** function enables IP notification events. Some events are used to indicate status, for example, if fax tone has been detected. Other events are used for Quality of Service (QoS) notifications on a particular media channel.

Notification events (solicited events) are different from asynchronous function termination events, such as IPMEV\_OPEN, which cannot be disabled. Once notification events are successfully enabled, if any of the specified events occur, the application is notified via SRL event management functions.

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

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pEvents</b>	<p>pointer to enumeration 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 – QoS alarm for excessive percentage of lost packets</li> <li>• EVT_JITTER – QoS alarm for excessive average jitter</li> <li>• EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity</li> <li>• EVT_RTPTIMEOUT – QoS alarm for RTP inactivity</li> <li>• EVT_FAXTONE – fax tone from TDM</li> <li>• EVT_RFC2833 – RFC 2833 event</li> <li>• EVT_T38CALLSTATE – T.38 call state event</li> </ul>

Parameter	Description
<b>unNumOfEvents</b>	number of events to enable
<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

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\_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_enbhdr( 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:** Asynchronous or synchronous (or both)

### ■ Description

The **ipm\_GetCTInfo( )** function returns information about a voice channel of an IPM device. This information is contained in a **CT\_DEVINFO** data structure.

Parameter	Description
<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_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

### ■ Cautions

This function will fail 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_ERRMSGP(chdev));
        exit(1);
    }

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

#### ■ See Also

- [ipm\\_Open\(\)](#)

## ipm\_GetLocalMediaInfo( )

**Name:** int ipm\_GetLocalMediaInfo(nDeviceHandle, \*pMediaInfo, usMode)

**Inputs:**

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

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
ipmlib.h

**Category:** Media Session

**Mode:** asynchronous or synchronous

### ■ Description

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

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
<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.
<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

To retrieve RTP or T.38 information, set the `eMediaType` field to `MEDIATYPE_LOCAL_RTP_INFO` or `MEDIATYPE_LOCAL_UDPTL_T38_INFO` and set `unCount` to 1. See the example for details.

## ■ 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_RTP_INFO;
    // MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_UDPTL_T38_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_LOCAL_RTP_INFO == pMediaInfo->MediaData[i].eMediaType)
                    printf("MediaType = MEDIATYPE_LOCAL_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:** srlib.h  
ipmlib.h

**Category:** System Control

**Mode:** asynchronous or synchronous

### ■ 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.

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, that is, the data structure [IPM\\_PARM\\_INFO](#) has been filled in. Use SRL functions to retrieve 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_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 = 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_getevtttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_PARM_INFO* pParmInfo;

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

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

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

```

## ■ See Also

- [ipm\\_SetParm\(\)](#)

## ipm\_GetQoSAlarmStatus( )

**Name:** int ipm\_GetQoSAlarmStatus(nDeviceHandle, \*pQoSAlarmInfo, usMode)

**Inputs:**

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

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
ipmlib.h

**Category:** QoS

**Mode:** asynchronous or synchronous

### ■ Description

The **ipm\_GetQoSAlarmStatus( )** function retrieves the ON/OFF state of all QoS alarms enumerated in eIPM\_QOS\_TYPE. Quality of Service (QoS) alarms report the status of a media channel, they do not report board-level alarms.

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

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media 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; that is, alarm status information was filled in. Use SRL functions to retrieve [IPM\\_QOS\\_ALARM\\_STATUS](#) structure fields.

IPMEV\_ERROR

Indicates that the function failed.

### ■ Cautions

The function returns the status of all QoS alarms that are enumerated in eIPM\_QOS\_TYPE.

## ■ 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 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.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:** srlib.h  
 ipmlib.h

**Category:** QoS

**Mode:** asynchronous or synchronous

### ■ Description

The **ipm\_GetQoSThreshold()** function retrieves QoS alarm threshold settings. Quality of Service (QoS) alarms report the status of media channels only; they do not report board-level alarms.

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; that is, alarm threshold settings were returned. 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.

## ■ 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 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\(\)](#)



## ■ Cautions

- The application can call [ipm\\_GetQoSAlarmStatus\(\)](#) to retrieve alarm information for the current session.

## ■ Errors

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

**EIPM\_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_enbhdr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

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

```

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

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

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

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

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

```

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

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

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

■ **See Also**

- [ipm\\_GetQoSAlarmStatus\(\)](#)
- [ipm\\_StartMedia\(\)](#)

## ipm\_GetXmitSlot()

**Name:** int ipm\_GetXmitSlot(nDeviceHandle, \*pTimeslotInfo, usMode)

**Inputs:**

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

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
ipmlib.h

**Category:** System Control

**Mode:** asynchronous or synchronous

### ■ 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; that is, the TDM time slot information data structure was filled in. 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

### ■ 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.

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.

**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.

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 <b>SC_TSINFO</b> for details.
<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\_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.

## ■ 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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
     *
     * Main Processing
     *
     */

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

```

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

/*
 *
 * Continue processing
 *
 */
}

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

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

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

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

```

## ■ See Also

- [\*\*ipm\\_UnListen\(\)\*\*](#)

## ipm\_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

### ■ 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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

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

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

```

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

void CheckEvent()
{
    int nDeviceID = sr_getevtddev();
    int nEventType = sr_getevtttype();

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

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

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

```

## ■ See Also

- [ipm\\_Close\(\)](#)

## ipm\_ReceiveDigits( )

**Name:** int ipm\_ReceiveDigits(nDeviceHandle, \*pDigitInfo, usMode)

**Inputs:**

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

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
ipmlib.h

**Category:** I/O

**Mode:** asynchronous or synchronous

### ■ Description

The **ipm\_ReceiveDigits( )** function enables the IP channel to receive digits from the IP network or 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 unNumberOfDigits field to indicate how many IPMEV_DIGITS_RECEIVED events the application must process. See <a href="#">IPM_DIGIT_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\_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 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*.
- Digits are only received if there is an active RTP session; if two ipm devices are directly routed together, you must establish an RTP session before digits can be sent and received.

## ■ Errors

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

EIPM\_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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

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

```

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

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

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

    /*
     *
     * Perform Error Processing
     *
     */
}

/*
 *
 * Continue processing
 *
 */
}

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

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

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

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

```



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

■ **See Also**

- [ipm\\_SendDigits\(\)](#)

## ipm\_ResetQoSAlarmStatus( )

**Name:** int ipm\_ResetQoSAlarmStatus(nDeviceHandle, \*pQoSAlarmInfo, usMode)

**Inputs:**

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

**Returns:** 0 on success  
-1 on failure

**Includes:** srllib.h  
ipmlib.h

**Category:** QoS

**Mode:** asynchronous or synchronous

### ■ Description

The **ipm\_ResetQoSAlarmStatus( )** function resets QoS alarm(s) to the OFF state. Quality of Service (QoS) alarms report the status of a media channel, they do not report board-level alarms.

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) have 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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
     *
     * Main Processing
     *
     */

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

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

void CheckEvent()
{
    int nEventType = sr_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

### ■ Description

The **ipm\_SendDigits( )** function generates the supplied digits in the specified direction.

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 16.
<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 `eIPM_DIGIT_DIRECTION` is to send digits toward the TDM bus.
- Digits are only sent if there is an active RTP session; if two ipm devices are directly routed together, you must establish an RTP session before digits can be sent and received.

## ■ Errors

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

`EIPM_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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
    .
    .
    Main Processing
    .
    .
    */

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

```

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

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

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

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

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

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

```

## ■ See Also

- [ipm\\_ReceiveDigits\(\)](#)

## ipm\_SendRFC2833SignalIDToIP( )

**Name:** int ipm\_SendRFC2833SignalIDToIP(nDeviceHandle, \* pSignalInfo, usMode)

**Inputs:**

int nDeviceHandle	• IP Media device handle
IPM_RFC2833_SIGNALID_INFO *pSignalInfo	• 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

### ■ Description

The **ipm\_SendRFC2833SignalIDToIP( )** function sends the supplied RFC 2833 signal to IP.

Parameter	Description
<b>nDeviceHandle</b>	handle of the IP Media device
<b>pSignalInfo</b>	pointer to structure that contains RFC 2833 signal ID and state information  See <a href="#">IPM_RFC2833_SIGNALID_INFO</a> for details.  Note that the application must fill in the RFC 2833 signal to be sent.
<b>usMode</b>	operation mode  Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

### ■ Termination Events

IPMEV\_SEND\_SIGNAL\_TO\_IP

Indicates successful completion; that is, the supplied RFC 2833 signal was sent. 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 <string.h>
#include <srllib.h>
#include <ipmlib.h>

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

void main()
{
    int nDeviceHandle;
    IPM_RFC2833_SIGNALID_INFO SignalInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
     *
     * Main Processing
     *
     */

    /*
     * Generate the start of an RFC2833 ringback packet to IP.
     * ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
     */
    SignalInfo.eSignalID = SIGNAL_ID_EVENT_LINE_RINGING_TONE;
    SignalInfo.eState = SIGNAL_STATE_ON;

    if(ipm_SendRFC2833SignalIDToIP(nDeviceHandle, &SignalInfo, EV_ASYNC) == -1)
    {
        printf("ipm_SendRFC2833SignalIDToIP failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
    }
    /*
     *
     */
}
```

```

        Perform Error Processing
        .
        .
        */
    }

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

    /*
    Generate the end of an RFC2833 ringback packet to IP.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    SignalInfo.eSignalID = SIGNAL_ID_EVENT_LINE_RINGING_TONE;
    SignalInfo.eState = SIGNAL_STATE_OFF;
    if(ipm_SendRFC2833SignalIDToIP(nDeviceHandle, &SignalInfo, EV_ASYNC) == -1)
    {
        printf("ipm_SendRFC2833SignalIDToIP failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
}

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

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

        //Successful reply to ipm_SendRFC2833SignalIDToIP()
        case IPMEV_SEND_SIGNAL_TO_IP:
            printf("Received IPMEV_SEND_SIGNAL_TO_IP 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

None.

## 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:** srlib.h  
ipmlib.h

**Category:** System Control

**Mode:** asynchronous or synchronous

### ■ 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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
     *
     * Main Processing
     *
     */

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

    /*
     *
     * . continue
     *
     */
}

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

```

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

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

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

```

#### ■ See Also

- [ipm\\_GetParm\(\)](#)

## ipm\_SetQoSThreshold( )

**Name:** int ipm\_SetQoSThreshold(nDeviceHandle, \*pInfo, 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

### ■ Description

The **ipm\_SetQoSThreshold( )** function changes QoS alarm threshold settings. Quality of Service (QoS) alarms report the status of a media channel, they do not report board-level alarms. Use this function to set the trigger levels for QoS alarms. This function can be called at any time, including when a session is in progress.

If **mode** is EV\_SYNC, the function returns 0 if successful; otherwise -1 is returned. The current QoS alarm identifier's settings are returned via the pointer to [IPM\\_QOS\\_THRESHOLD\\_INFO](#).

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\_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

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;
    IPM_QOS_THRESHOLD_INFO mySetQoSThresholdInfo;
    // Register event handler function with srl
    sr_enbhdrlr( 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

---

### ■ 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> </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 – receives data from the IP network but no data is sent</li> <li>• DATA_IP_SENDOONLY – sends data to the IP network but no data is received</li> <li>• DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM; used for gateway functionality.</li> <li>• DATA_MULTICAST_SERVER – multicast server mode</li> <li>• DATA_MULTICAST_CLIENT – multicast client mode</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 93 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_REMOTE_RTP_INFO;
    MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
    strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.9\n");

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

    MediaInfo.MediaData[2].eMediaType = MEDIATYPE_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_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( )

**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

### ■ 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 – receives data from the IP network but no data is sent</li> <li>• DATA_IP_SENDOONLY – sends data to the IP network but no data is received</li> <li>• DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM; used for gateway functionality</li> <li>• DATA_MULTICAST_SERVER – multicast server mode</li> <li>• DATA_MULTICAST_CLIENT – multicast client mode</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.

### IPMEV\_ERROR

Indicates that the function failed.

## ■ Cautions

- The application must wait until this function completes before calling [ipm\\_Listen\( \)](#).
- 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_REMOTE_RTP_INFO;
    MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
    strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.9\n");

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

    MediaInfo.MediaData[2].eMediaType = MEDIATYPE_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_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_getevtddev();
    int nEventType = sr_getevtttype();

    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\\_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

### ■ 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>	media operation enumeration; 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 – operation of receiving digits</li> <li>• STOP_MEDIA – operation of media 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 eIPM\_STOP\_OPERATION 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_enbhdlr( EV_ANYDEV ,EV_ANYEVT , (HDLR)CheckEvent);

    /*
     *
     * . Main Processing
     *
     */

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

```

    /*
    .
    . Continue Processing
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_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

---

### ■ Description

The **ipm\_UnListen( )** function stops listening to the TDM time slot specified in a previous call to **ipm\_Listen( )**.

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

Parameter	Description
<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.

## ■ 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\\_enbhdr\( \)](#) 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. IPM\_DIGIT\_INFO contains data. One event is returned for each digit that is received.

## IPMEV\_ERROR

Termination event. No data is returned. Event generated on any handle when there is an error.

## IPMEV\_EVENT\_DISABLED

Termination event for [ipm\\_DisableEvents\( \)](#). No data is returned. Indicates specified IP notification events have been disabled.

## IPMEV\_EVENT\_ENABLED

Termination event for [ipm\\_EnableEvents\( \)](#). No data is returned. Indicates specified IP notification events have been enabled.

## IPMEV\_FAXTONE

Unsolicited event for [ipm\\_EnableEvents\( \)](#). IPM\_FAX\_SIGNAL contains data. Event is returned when fax tone is detected on TDM.

## IPMEV\_GET\_LOCAL\_MEDIA\_INFO

Termination event for [ipm\\_GetLocalMediaInfo\( \)](#). IPM\_MEDIA\_INFO contains data. Indicates local media information has been returned.

## IPMEV\_GET\_PARM

Termination event for [ipm\\_GetParm\( \)](#). IPM\_PARM\_INFO contains data. Indicates IP channel parameters have been returned.

## IPMEV\_GET\_QOS\_ALARM\_STATUS

Termination event for [ipm\\_GetQoSAlarmStatus\( \)](#). IPM\_QOS\_ALARM\_STATUS contains data. Indicates alarm status information was filled in.

**IPMEV\_GET\_QOS\_THRESHOLD\_INFO**

Termination event for [ipm\\_GetQoSThreshold\(\)](#). IPM\_QOS\_THRESHOLD\_INFO contains data. Indicates alarm threshold settings have been returned.

**IPMEV\_GET\_SESSION\_INFO**

Termination event for [ipm\\_GetSessionInfo\(\)](#). IPM\_SESSION\_INFO contains data. Indicates statistics for previous session have been returned.

**IPMEV\_GET\_XMITTS\_INFO**

Termination event for [ipm\\_GetXmitSlot\(\)](#). SC\_TSINFO contains data. Indicates TDM time slot information has been returned.

**IPMEV\_LISTEN**

Termination event for [ipm\\_Listen\(\)](#). No data is returned. Indicates time slot routing was successfully completed.

**IPMEV\_OPEN**

Termination event for [ipm\\_Open\(\)](#). No data is returned. Indicates IP channel was successfully opened and device handle is valid.

**IPMEV\_QOS\_ALARM**

Unsolicited event for [ipm\\_EnableEvents\(\)](#). No data is returned. Event is returned when desired QoS alarm triggers.

**IPMEV\_RECEIVE\_DIGITS**

Termination event for [ipm\\_ReceiveDigits\(\)](#). No data is returned. Indicates channel has been enabled to receive digits.

**Note:** IPMEV\_DIGITS\_RECEIVED indicates digit transfer has occurred.

**IPMEV\_RESET\_QOS\_ALARM\_STATUS**

Termination event for [ipm\\_ResetQoSAlarmStatus\(\)](#). No data is returned. Indicates specified QoS alarms have been reset to OFF state.

**IPMEV\_RFC2833SIGNALRECEIVED**

Unsolicited event for [ipm\\_EnableEvents\(\)](#). IPM\_RFC2833\_SIGNALID\_INFO contains data. Event is returned when RFC 2833 signal is detected on IP.

**IPMEV\_SEND\_DIGITS**

Termination event for [ipm\\_SendDigits\(\)](#). No data is returned. Indicates supplied digits were sent successfully.

**IPMEV\_SEND\_SIGNAL\_TO\_IP**

Termination event for [ipm\\_SendRFC2833SignalIDToIP\(\)](#). No data is returned. Indicates RFC2833 message has been sent to IP.

**IPMEV\_SET\_PARM**

Termination event for [ipm\\_SetParm\(\)](#). No data is returned. Indicates IP channel parameters have been modified.

**IPMEV\_SET\_QOS\_THRESHOLD\_INFO**

Termination event for [ipm\\_SetQoSThreshold\(\)](#). IPM\_QOS\_THRESHOLD\_INFO contains data. Indicates modified QoS alarm threshold levels have been returned.

**IPMEV\_STARTMEDIA**

Termination event for [ipm\\_StartMedia\(\)](#). No data is returned. Indicates media channel information has been set and session has been started.

**IPMEV\_STOPPED**

Termination event for [ipm\\_Stop\(\)](#). No data is returned. Indicates all on-going activity on the IP channel has terminated.

**IPMEV\_T38CALLSTATE**

Unsolicited event for [ipm\\_EnableEvents\(\)](#). eIPM\_T38CALLSTATE contains data. Event is returned when T.38 call state changes.

**IPMEV\_UNLISTEN**

Termination event for [ipm\\_UnListen\(\)](#). No data is returned. Indicates IP channel was disconnected from TDM time slot.



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 .....	90
• IPM_CLOSE_INFO .....	92
• IPM_CODER_INFO .....	93
• IPM_DIGIT_INFO .....	95
• IPM_EVENT_INFO .....	96
• IPM_FAX_SIGNAL .....	97
• IPM_MEDIA .....	98
• IPM_MEDIA_INFO .....	99
• IPM_OPEN_INFO .....	100
• IPM_PARM_INFO .....	101
• IPM_PORT_INFO .....	102
• IPM_QOS_ALARM_DATA .....	103
• IPM_QOS_ALARM_STATUS .....	104
• IPM_QOS_SESSION_INFO .....	105
• IPM_QOS_THRESHOLD_DATA .....	106
• IPM_QOS_THRESHOLD_INFO .....	108
• IPM_RFC2833_SIGNALID_INFO .....	109
• IPM_RTCP_SESSION_INFO .....	111
• IPM_SESSION_INFO .....	113
• SC_TSINFO .....	114

## 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 Dialogic API libraries where many additional values may be used.

### ■ Field Descriptions

The fields of the CT\_DEVINFO data structure are described as follows:

**ct\_prodid**

Contains a valid product identification number for the device.

**ct\_devfamily**

Specifies the device family. Possible values are:

- CT\_DFHMPDM3 – HMP (Host Media Processing) device

**ct\_devmode**

Specifies the device mode. Possible values are:

- CT\_DMNETWORK – DM3 network device

**ct\_nettype**

Specifies the type of network interface for the device. Possible values are:

- CT\_NTIPT – IP connectivity

**ct\_busmode**

Specifies the bus architecture used to communicate with other devices in the system. Possible values are:

- CT\_BMSCBUS – TDM bus architecture

ct\_busencoding

Describes the PCM encoding used on the bus. Possible values are:

- CT\_BEULAW – mu-law encoding
- CT\_BEALAW – A-law encoding

ct\_ext\_devinfo.ct\_RFU

Not used.

ct\_ext\_devinfo.ct\_net\_devinfo.ct\_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;          /* Flag indicating if VAD is */
                                                /* enabled/disabled */
    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\\_SetRemoteMediaInfo\(\)](#), and [ipm\\_StartMedia\(\)](#) functions.

Table 1 shows the coders and properties that are supported when using the IP Media Library with Host Media Processing (HMP) software.

### Field Descriptions

The fields of the IPM\_CODER\_INFO data structure are described as follows. Refer to Table 1 for coder-specific guidelines for filling in these fields.

#### eCoderType

type of coder to be used for streaming media operations. .

The following values are defined:

- CODER\_TYPE\_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\_G7231\_6\_3K – G.723.1, 6.3 kbps
- CODER\_TYPE\_G729ANNEXA – G.729 Annex A
- CODER\_TYPE\_G729ANNEXAWANNEXB – G.729 Annex A with Annex B

#### eFrameSize

size of frame for coders that support multiple frame sizes—currently G.711 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\_FRAME\_SIZE to specify that value.

The eIPM\_CODER\_FRAME\_SIZE data type is an enumeration which specifies the frame size for G.711 coders only. The following values for eIPM\_CODER\_FRAME\_SIZE are enumerated:

- CODER\_FRAME\_SIZE\_10 – frame size = 10 ms
- CODER\_FRAME\_SIZE\_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 1. 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 1890 standard definitions. The application is responsible for negotiating this value between the two endpoints. This may be set to any value for non-standard coders or if the application does not require interoperability with third-party applications. Values: 0 to 127. 96 to 127 is the dynamic range.

#### 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 1. Supported Coder Properties for Host Media Processing**

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
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_G723_1_5_3K	(fixed at 30)	2 or 3	Either value
CODER_TYPE_G723_1_6_3K	(fixed at 30)	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_ENABLE <sup>1</sup>
<b>NOTES:</b> 1. Applications must explicitly enable VAD even though G.723a+b implicitly supports VAD.			

## 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 IP network and 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**

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**

set to DIGIT\_TDM

The eIPM\_DIGIT\_DIRECTION data type is an enumeration which identifies the direction of digit flow. The enumeration defines the following value:

- DIGIT\_TDM – digits are sent to or received from the TDM bus

**cDigits[MAX\_IPM\_DIGITS]**

when sending digits, the actual digits to be sent; not used when receiving digits

**unNumberOfDigits**

number of digits being sent or received. When sending digits via [ipm\\_SendDigits\(\)](#), this field indicates the number of digits to be sent; the maximum number of digits that may be sent is 16. When receiving digits via [ipm\\_ReceiveDigits\(\)](#), upon return the function sets this field to the actual number of digits to be received via asynchronous events.

**unTimeStamp**

reserved for future use; set to 0

**unExpirationTime**

reserved for future use; set to 0

**unDuration**

reserved for future use; set to 0

## IPM\_EVENT\_INFO

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

### ■ Description

This structure is used for IP event notification. See [Chapter 3, “Events”](#) for more information.

### ■ Field Descriptions

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

**unCount**  
number of data structures pointed to

**\*pEventData**  
pointer to structure containing event-specific data

## IPM\_FAX\_SIGNAL

```
typedef struct sc_tsinfo {  
    eIPM_TONE eToneType;  
    unsigned int unToneDuration;  
  
} IPM_FAX_SIGNAL, *PIPM_FAX_SIGNAL;
```

### ■ Description

This structure defines the tone information detected by the gateway. IPM\_FAX\_SIGNAL is a child of IPM\_MEDIA, which is a child of the IPM\_MEDIA\_INFO structure. The structure is used by the [ipm\\_GetLocalMediaInfo\(\)](#) and [ipm\\_StartMedia\(\)](#) functions (and the deprecated [ipm\\_SetRemoteMediaInfo\(\)](#) function).

### ■ Field Descriptions

The fields of the IPM\_FAX\_SIGNAL data structure are described as follows:

#### eToneType

The eIPM\_TONE data type is an enumeration which defines the following tone types:

- 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;       /* 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](#), and [IPM\\_FAX\\_SIGNAL](#). This structure is a child of the [IPM\\_MEDIA\\_INFO](#) structure which is used by the [ipm\\_GetLocalMediaInfo\(\)](#) and [ipm\\_StartMedia\(\)](#) functions (and the deprecated [ipm\\_SetRemoteMediaInfo\(\)](#) function).

### ■ Field Descriptions

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

#### eMediaType

type of media used to start 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\_LOCAL\_CODER\_INFO – local receive coder information
- MEDIATYPE\_LOCAL\_RTCP\_INFO – local RTCP port information
- MEDIATYPE\_LOCAL\_RTP\_INFO – local RTP port information
- MEDIATYPE\_LOCAL\_UDPTL\_T38\_INFO – local UDP packet T.38 information
- MEDIATYPE\_REMOTE\_CODER\_INFO – remote receive coder information
- MEDIATYPE\_REMOTE\_RTCP\_INFO – remote RTCP port information
- MEDIATYPE\_REMOTE\_RTP\_INFO – remote RTP port information
- MEDIATYPE\_REMOTE\_UDPTL\_T38\_INFO – remote UDP packet T.38 information

#### PortInfo

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

#### CoderInfo

reference to coder information structure [IPM\\_CODER\\_INFO](#)

#### FaxSignal

reference to fax signal structure [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\(\)](#) and [ipm\\_StartMedia\(\)](#) functions (and the deprecated [ipm\\_SetRemoteMediaInfo\(\)](#) function).

### ■ 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 2 for values.

pvParmValue

pointer to the value of the parameter

**Table 2. eIPM\_PARM Parameters and Values**

eIP_PARM Define	Description and Values
PARMCH_DTMFXFERMODE	sets DTMF transfer mode. Type: eIPM_DTMFXFERMODE. Values: <ul style="list-style-type: none"> <li>DTMF_XFERMODE_INBAND – in-band (default)</li> <li>DTMF_XFERMODE_OUTOFBAND – out-of-band</li> <li>DTMF_XFERMODE_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_RFC2833EVT_RX_PLT	RFC2833 event receive payload. Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMCH_RFC2833EVT_TX_PLT	RFC2833 event transmit payload. Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMCH_RX_ADJVOLUME	set volume level adjustment, in dB, for inbound (from IP) side. Type: integer. Range: -32 to 31. Default: 0 (no adjustment).
PARMCH_TX_ADJVOLUME	set volume level adjustment, in dB, for outbound (to IP) side. Type: integer. Range: -32 to 31. Default: 0 (no adjustment).
PARMCH_TOS	set type of service. Type: char. Range: 0 to 255. Default: 0.
PARMCH_TTL	set time-to-live for multicast. Type: char. Range: 0 to 255. Default: 1.

## IPM\_PORT\_INFO

```
typedef struct ipm_port_info_tag
{
    unsigned int    unPortId;           /* The Port ID */
    char            cIPAddress[IP_ADDR_SIZE]; /* IP Address */

} IPM_PORT_INFO, *PIPM_PORT_INFO;
```

### ■ Description

This structure contains RTP, RTCP, and T.38 UDP port properties. It is a child of [IPM\\_MEDIA](#), which is a child of the [IPM\\_MEDIA\\_INFO](#) structure. The structure is used by the [ipm\\_GetLocalMediaInfo\(\)](#) and [ipm\\_StartMedia\(\)](#) functions.

### ■ 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. It is a child of the [IPM\\_QOS\\_ALARM\\_STATUS](#) structure which is used by [ipm\\_GetQoSAlarmStatus\(\)](#) and [ipm\\_ResetQoSAlarmStatus\(\)](#).

ForHMP, the software generates a QoS 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.

### ■ Field Descriptions

The fields of the IPM\_QOS\_ALARM\_DATA data structure are described as follows:

eQoSType

identifies the QoS alarm event that has occurred

The eIPM\_QOS\_TYPE data type is an enumeration which defines the following values:

- QOSTYPE\_LOSTPACKETS – excessive lost packets
- QOSTYPE\_JITTER – excessive average jitter
- QOSTYPE\_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE\_RTPTIMEOUT – RTP inactivity

eAlarmState

alarm on / off flag

The eIPM\_ALARM\_STATE data type is an enumeration which defines the following values:

- ALARM\_STATE\_OFF – QoS alarm is OFF
- ALARM\_STATE\_ON – QoS 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\_LOSTPACKETS – percent of lost packets since the beginning of the call
- QOSTYPE\_JITTER – average jitter since the beginning of the call (in msec)
- QOSTYPE\_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE\_RTPTIMEOUT – RTP inactivity

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 3 will be used unless [ipm\\_SetQoSThreshold\(\)](#) is used to set non-default values. Note that the application must explicitly set all applicable values in the IPM\_QOS\_THRESHOLD structure even when default values are desired.

### 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\_LOSTPACKETS – percent of lost packets since the beginning of the call.
- QOSTYPE\_JITTER – average jitter since the beginning of the call (in msec).
- QOSTYPE\_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE\_RTPTIMEOUT – RTP inactivity

#### unTimeInterval

time interval (in ms) between successive parameter measurements. Value should be set to a multiple of 100.

**Note:** Value must not be greater than unFaultThreshold for QoS parameter types that have a time-based threshold.

#### unDebounceOn

time interval for detecting potential alarm fault condition. Must be set to a value that is a multiple of unTimeInterval.

**Note:** This field is not used for RTCP and RTP Timeout alarms and must be set to 0.

#### unDebounceOff

time interval for detecting potential alarm non-fault condition. Must be set to a value that is a multiple of unTimeInterval.

**Note:** This field is not used for RTCP and RTP Timeout alarms and must be set to 0.

#### unFaultThreshold

fault threshold parameter. The meaning and value range of this field depends on the QoS Type:

- QOSTYPE\_LOSTPACKET allowable percentage of lost packets. Range: 0 to 100 (%)
- QOSTYPE\_JITTER allowable average jitter, in ms. Range: 0 to 1000 (ms)
- QOSTYPE\_RTCPTIMEOUT – allowable inactive interval before an alarm is sent, in units of 100 ms. Range: 50 to 1200 (x100 ms)
- QOSTYPE\_RTPTIMEOUT – allowable inactive interval before an alarm is sent, in units of 100 ms units. Range: 50 to 1200 (x100 ms)

#### unPercentSuccessThreshold

percentage of poll instances in unDebounceOff time interval that the fault threshold must not be exceeded before an “alarm off” event is sent

**Note:** This parameter is not used for RTCP and RTP Timeout alarms and must be set to 0.

#### unPercentFailThreshold

percentage of poll instances in unDebounceOn time interval that the fault threshold must be exceeded before an “alarm on” event is sent

**Note:** This parameter is not used for RTCP and RTP Timeout alarms and must be set to 0.

**Table 3. Quality of Service Parameter Defaults for Host Media Processing**

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold <sup>1</sup>	% Success Threshold	% Fail Threshold
Lost Packets	1000	10000	10000	20 (%)	40	40
Jitter	5000	20000	60000	60 (ms)	25	25
RTCP Timeout	1000	0	0	250 (x100ms = 25sec)	0	0
RTP Timeout	1000	0	0	1200 (x100ms = 120sec)	0	0
Notes: 1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.						

## 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 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 sends RFC 2833-compliant signal IDs and states. It is used by the [ipm\\_SendRFC2833SignalIDToIP\(\)](#) function.

### Field Descriptions

The fields of the IPM\_RFC2833\_SIGNALID\_INFO data structure are described as follows:

**eSignalID**

signal ID to send

The eIPM\_RFC2833\_SIGNAL\_ID data type enumeration defines values that are listed in Table 4.

**eState**

indicates whether the signal (tone) is on or off.

The eIPM\_SIGNAL\_STATE data type is an enumeration which defines the following values:

- SIGNAL\_STATE\_OFF – Signal is OFF; no tone is sent
- SIGNAL\_STATE\_ON – Signal is ON; tone is sent

There is no default state for eState; a value must be set by the application.

**Table 4. eIPM\_RFC2833\_SIGNAL\_ID Values**

Name	Value (H)
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

Table 4. eIPM\_RFC2833\_SIGNAL\_ID Values (Continued)

Name	Value (H)
SIGNAL_ID_EVENT_DTMF_B	0xd
SIGNAL_ID_EVENT_DTMF_C	0xe
SIGNAL_ID_EVENT_DTMF_D	0xf

## IPM\_RTCP\_SESSION\_INFO

```
typedef struct ipm_rtcp_session_info_tag
{
    unsigned int    unLocalSR_TimeStamp;
    unsigned int    unLocalSR_TxPackets;
    unsigned int    unLocalSR_TxOctets;
    unsigned int    unLocalSR_SendIndication;
    unsigned int    unLocalRR_FractionLost;
    unsigned int    unLocalRR_CumulativeLost;
    unsigned int    unLocalRR_SeqNumber;
    unsigned int    unLocalRR_ValidInfo;
    unsigned int    unRemoteSR_TimeStamp;
    unsigned int    unRemoteSR_TxPackets;
    unsigned int    unRemoteSR_TxOctets;
    unsigned int    unRemoteSR_SendIndication;
    unsigned int    unRemoteRR_FractionLost;
    unsigned int    unRemoteRR_CumulativeLost;
    unsigned int    unRemoteRR_SeqNumber;
    unsigned int    unRemoteRR_ValidInfo;
} IPM_RTCP_SESSION_INFO, *PIPM_RTCP_SESSION_INFO;
```

### ■ Description

This structure contains RTCP information for the session. It is a child of the [IPM\\_SESSION\\_INFO](#) structure which is filled in when [ipm\\_GetSessionInfo\(\)](#) returns successfully.

### ■ Field Descriptions

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

- unLocalSR\_TimeStamp  
time stamp of the RTCP packet transmission from the local sender
- unLocalSR\_TxPackets  
number of packets sent by the local sender
- unLocalSR\_TxOctets  
number of bytes sent by the local sender
- unLocalSR\_SendIndication  
local sender report has changed since the last transmission. Values may be either:
  - FALSE
  - TRUE
- unLocalRR\_FractionLost  
percentage of packets lost, as computed by the local receiver
- unLocalRR\_CumulativeLost  
number of packets lost, as computed by the local receiver
- unLocalRR\_SeqNumber  
last sequence number received from the local receiver
- unLocalRR\_ValidInfo  
reserved for future use

unRemoteSR\_TimeStamp  
time stamp of the RTCP packet transmission from the remote sender

unRemoteSR\_TxPackets  
number of packets sent by the remote sender

unRemoteSR\_TxOctets  
number of bytes sent by the remote sender

unRemoteSR\_SendIndication  
remote sender report has changed since the last transmission. Values may be either:

- FALSE
- TRUE

unRemoteRR\_FractionLost  
percentage of packets lost, as computed by the remote receiver

unRemoteRR\_CumulativeLost  
number of packets lost, as computed by the remote receiver

unRemoteRR\_SeqNumber  
last sequence number received from the remote receiver

unRemoteRR\_ValidInfo  
reserved for future use

## IPM\_SESSION\_INFO

```
typedef struct ipm_session_info_tag
{
    IPM_RTCP_SESSION_INFO  RtcpInfo;
    unsigned int           unQoSInfoCount;
    IPM_QOS_SESSION_INFO   QoSInfo[MAX_QOS_SESSION];
} IPM_SESSION_INFO, *PIPM_SESSION_INFO;
```

### ■ Description

This structure is a parent structure of the [IPM\\_RTCP\\_SESSION\\_INFO](#) and [IPM\\_QOS\\_SESSION\\_INFO](#) structures, and it is used by the [ipm\\_GetSessionInfo\( \)](#) function. It reports QoS statistics during the last IP session, including RTCP information. Note that it does not contain statistics for the current IP session.

### ■ Field Descriptions

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

RtcpInfo

reference to RTCP session information structure [IPM\\_RTCP\\_SESSION\\_INFO](#)

unQoSInfoCount

number of 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\_SetRemoteMediaInfo()**, **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\_SetRemoteMediaInfo()**, **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\\_SetRemoteMediaInfo\(\)](#), [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\\_SetRemoteMediaInfo\(\)](#), [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\\_SetRemoteMediaInfo\(\)](#), [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 2543. 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  
     HMP software 109  
 coder support, HMP software 94  
 coder types 93  
 conventions, device names 49  
 CT\_DEVINFO data structure 90

## D

data structures  
     CT\_DEVINFO 90  
     IPM\_CLOSE\_INFO 92  
     IPM\_CODER\_INFO 93  
     IPM\_DIGIT\_INFO 95  
     IPM\_EVENT\_INFO 96  
     IPM\_MEDIA 98  
     IPM\_MEDIA\_INFO 99  
     IPM\_OPEN\_INFO 100  
     IPM\_PORT\_INFO 102  
     IPM\_QOS\_ALARM\_DATA 103  
     IPM\_QOS\_ALARM\_STATUS 104  
     IPM\_QOS\_SESSION\_INFO 105  
     IPM\_QOS\_THRESHOLD\_DATA 106  
     IPM\_QOS\_THRESHOLD\_INFO 108  
     IPM\_RTCP\_SESSION\_INFO 111  
     IPM\_SESSION\_INFO 113  
     IPM\_TIMESLOT\_INFO 114  
     SC\_TSINFO 114

## I

I/O Functions 12  
     ipm\_ReceiveDigits() 52  
     ipm\_SendDigits() 59  
     ipm\_SendRFC2833SignalIDToIP() 62  
 ipm\_Close() 14  
 IPM\_CLOSE\_INFO data structure 92  
 IPM\_CODER\_INFO data structure 93  
 IPM\_DIGIT\_INFO data structure 95  
 ipm\_DisableEvents() 16  
 ipm\_EnableEvents() 20  
 IPM\_EVENT\_INFO data structure 96  
 IPM\_FAX\_SIGNAL data structure 97  
 ipm\_GetCTInfo() 24

ipm\_GetLocalMediaInfo() 26  
 ipm\_GetParm() 29  
 ipm\_GetQoSAlarmStatus() 32  
 ipm\_GetQoSThreshold() 35  
 ipm\_GetSessionInfo() 39  
 ipm\_GetXmitSlot() 43  
 ipm\_Listen() 46  
 IPM\_MEDIA data structure 98  
 IPM\_MEDIA\_INFO data structure 99  
 ipm\_Open() 49  
 IPM\_OPEN\_INFO data structure 100  
 IPM\_PORT\_INFO data structure 102  
 IPM\_QOS\_ALARM\_DATA data structure 103  
 IPM\_QOS\_ALARM\_STATUS data structure 104  
 IPM\_QOS\_SESSION\_INFO data structure 105  
 IPM\_QOS\_THRESHOLD\_DATA data structure 106  
 IPM\_QOS\_THRESHOLD\_INFO data structure 108  
 ipm\_ReceiveDigits() 52  
 ipm\_ResetQoSAlarmStatus() 56  
 IPM\_RTCP\_SESSION\_INFO data structure 111  
 ipm\_SendDigits() 59  
 ipm\_SendRFC2833SignalIDToIP() 62  
 IPM\_SESSION\_INFO data structure 113  
 ipm\_SetParm() 65  
 ipm\_SetQoSThreshold() 68  
 ipm\_SetRemoteMediaInfo() 71  
 ipm\_StartMedia() 75  
 ipm\_Stop() 79  
 IPM\_TIMESLOT\_INFO data structure 114  
 ipm\_UnListen() 82  
 IPMEV\_DIGITS\_RECEIVED event 53

## M

Media Session Functions 12  
     ipm\_GetCTInfo() 24  
     ipm\_GetLocalMediaInfo() 26  
     ipm\_GetSessionInfo() 39  
     ipm\_StartMedia() 75  
     ipm\_Stop() 79

## N

naming convention, devices 49

## Q

QoS Functions 12

- ipm\_GetQoSAlarmStatus( ) 32
- ipm\_GetQoSThreshold( ) 35
- ipm\_ResetQoSAlarmStatus( ) 56
- ipm\_SetQoSThreshold( ) 68

## S

SC\_TSINFO data structure 114

System Control Functions 11

- ipm\_Close( ) 14
- ipm\_DisableEvents( ) 16
- ipm\_EnableEvents( ) 20
- ipm\_GetParm( ) 29
- ipm\_GetXmitSlot( ) 43
- ipm\_Listen( ) 46
- ipm\_Open( ) 49
- ipm\_SetParm( ) 65
- ipm\_UnListen( ) 82