IP Media Library API for Linux and Windows Operating Systems

Library Reference

November 2002

05-1833-002

Publication Date: November 2002

Document Number: 05-1833-002

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 Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices** and other contact information, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Contents

Tables

About This Publication

The following topics provide information about this publication:

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.

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 Software 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 User's Guide*
- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *Standard Runtime Library API Library Reference*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/network/csp/* (for product information)

Function Summary by Category 1

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.

1.1 System Control Functions

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

ipm_Close( )
closes an IP channel

ipm_DisableEvents( )
disables IP notification events

ipm_EnableEvents( )
enables IP notification events

ipm_GetParm( )
returns IP channel parameters

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

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

ipm_Open( )
opens an IP channel and returns a handle

ipm_Ping( )
generates a message to a remote IP address

ipm_SetParm( )
sets IP channel parameters

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

1.2 I/O (Input/Output) Functions

The following functions are used to transfer digits and data:

ipm_ReceiveDigits( )
enables the IP channel to receive digits from the 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_GetLocalMediaInfo( )
retrieves properties for the local media channel

ipm_GetSessionInfo( )
retrieves statistics for the current session

ipm_SetRemoteMediaInfo( )
sets media properties and starts the session

Note: This function is not recommended; 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

1.5 IP Media Function Support by Platform

Table 1, "IP Media Function Support by Platform", on page 11 provides an alphabetical listing of IP media API functions. The table indicates which platforms are supported for each of the functions. There are three platforms that use the IP media library:

Intel® NetStructure™ DM/IP Series boards
: These boards feature 24–60 ports-per-slot of both public network and Internet connectivity plus onboard voice, fax, and speech processing. The boards are scalable to support access gateways, IP-PBXs, and media server applications.

Intel® NetStructure™ IPT Series boards
: These boards provide high-density, standards-based VOIP interface boards for developing scalable, carrier-grade IP telephony gateways and media servers.

Intel® NetStructure™ Host Media Processing (HMP) software
: The HMP software performs voice, conferencing and IVR processing on general-purpose servers based on Intel® architecture without the use of specialized hardware.

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

Table 1. IP Media Function Support by Platform

Function	DM/IP Boards	IPT Boards	HMP Software
ipm_Close( )	S	S	S
ipm_DisableEvents( )	S	S	S
ipm_EnableEvents( )	S	S	S
ipm_GetLocalMediaInfo( )	S	S	S
ipm_GetParm( )	S	S	S
ipm_GetQoSAlarmStatus( )	S	NS	S
ipm_GetQoSThreshold( )	S	S†	S
ipm_GetSessionInfo( )	S	NS	S
ipm_GetXmitSlot( )	S	S	S
ipm_Listen( )	S	S	S
ipm_Open( )	S	S	S
ipm_Ping( )	NS	S	NS
ipm_ReceiveDigits( )	S	S	S
ipm_ResetQoSAlarmStatus( )	S	NS	S
ipm_SendDigits( )	S	S	NS
ipm_SendRFC2833SignalIDToIP( )	S	NS	S
ipm_SetParm( )	S	S	S

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

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

Function	DM/IP Boards	IPT Boards	HMP Software
ipm_SetQoSThreshold( )	S	S†	S
ipm_SetRemoteMediaInfo( )	S	S	S
ipm_StartMedia( )	S	S	S
ipm_Stop( )	S	S	S
ipm_UnListen( )	S	S	S
Legend: NS = Not Supported, S = Supported, † = Variance between platforms, refer to Function Description for more information.			

Function Information 2

This chapter contains a detailed description of each 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

Platform: DM/IP, IPT, HMP

Description

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

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

Termination Events

None - this function operates in synchronous mode only.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_CONFIG
Configuration error

EIPM_FWERROR
Firmware error

Example

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

void main()
{
    int nDeviceHandle;
    /*
    .
    Main Processing
    .
    .
    .
    */

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

See Also

- **ipm_Open( )**

ipm_DisableEvents( )

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

Inputs:

int nDeviceHandle	• IP Media device handle
eIPM_EVENT *pEvents	• 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

Platform: DM/IP, IPT, HMP

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
nDeviceHandle	handle of the IP Media device
pEvents	pointer to enumeration that specifies the events to disable. *Note:* EVT_DTMFDISCARDED is not supported on Intel® NetStructure™ IPT Series boards. EVT_ROUNDTRIPLATENCY is not supported on Intel® NetStructure™ DM/IP Series boards. The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_DTMFDISCARDED – number of lost DTMF digits since the beginning of the call • EVT_LOSTPACKETS – percent of lost packets since the beginning of the call • EVT_JITTER – average jitter since the beginning of the call (in msec) • EVT_ROUNDTRIPLATENCY – RTP packet latency • EVT_FAXTONE – fax tone from TDM • EVT_RFC2833 – RFC 2833 events • EVT_T38FAXTONE – fax tone message from T.38 UDP packet • EVT_T38CALLSTATE – T.38 call state events
unNumOfEvents	number of events to disable
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
: indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
: Invalid parameter

EIPM_INTERNAL
: Internal error

EIPM_INV_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[3] ={EVT_DTMFDISCARDED, EVT_LOSTPACKETS, EVT_JITTER};
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Application is shutting down
    Need to disable all enabled events for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open() and
    The events listed in myEvents were enabled sometime earlier.
    */
    if(ipm_DisableEvents(nDeviceHandle, myEvents, 3, EV_ASYNC) == -1)
    {
        printf("ipm_DisableEvents failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
    /*
    .
    Continue shut down
    .
    */
}
```

```
void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
        /*
        .
        . Other events
        .
        */
        /* Expected reply to ipm_DisableEvents */
        case IPMEV_EVENT_DISABLED:
            printf("Received IPMEV_EVENT_DISABLED for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;
        default:
            printf("Received unknown event = %d for device = %s\n",
                nEventType, ATDV_NAMEP(nDeviceID));
            break;
    }
}
```

See Also

- ipm_EnableEvents( )

ipm_EnableEvents( )

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

Inputs:

int nDeviceHandle	• IP Media device handle
eIPM_EVENT *pEvents	• 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: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

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.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pEvents	pointer to enumeration that specifies the events to enable *Note:* EVT_DTMFDISCARDED is not supported on Intel® NetStructure™ IPT Series boards. EVT_ROUNDTRIPLATENCY is not supported on Intel® NetStructure™ DM/IP Series boards. The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_DTMFDISCARDED – number of lost DTMF digits since the beginning of the call • EVT_LOSTPACKETS – percent of lost packets since the beginning of the call • EVT_JITTER – average jitter since the beginning of the call (in msec) • EVT_ROUNDTRIPLATENCY – RTP packet latency • EVT_FAXTONE – fax tone from TDM • EVT_RFC2833 – RFC 2833 events • EVT_T38FAXTONE – fax tone message from T.38 UDP packet • EVT_T38CALLSTATE – T.38 call state events
unNumOfEvents	number of events to enable
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_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[3] ={EVT_DTMFDISCARDED, EVT_LOSTPACKETS, EVT_JITTER};
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Need to enable three events for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_EnableEvents(nDeviceHandle, myEvents, 3, EV_ASYNC) == -1)
    {
        printf("ipm_EnableEvents failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
    /*
    .
    . Continue Processing
    .
    */
}
```

```
void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
        /*
        .
        . List of expected events
        .
        */
        /* Expected reply to ipm_EnableEvents() */
        case IPMEV_EVENT_ENABLED:
            printf("Received IPMEV_EVENT_ENABLED for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            break;
        default:
            printf("Received unknown event = %d for device = %s\n",
                nEventType, ATDV_NAMEP(nDeviceID));
            break;
    }
}
```

See Also

- ipm_DisableEvents( )

ipm_GetLocalMediaInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

- To retrieve RTP or T.38 information, set the eMediaType field to MEDIATYPE_RTP_INFO or MEDIATYPE_T38_INFO and set unCount to 1. See the example for details.
- When using Intel® NetStructure™ IPT Series boards, the following limitations apply:
  - For a non-load balancing configuration, if this function is called multiple times, it could return a different port number for a specified channel.
  - In load balancing mode, if this function is called multiple times, it could return a different IP/Port pair each time.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

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

- **ipm_SetRemoteMediaInfo( )**

ipm_GetParm( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

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

Termination Events

IPMEV_GET_PARM
indicates successful completion, that is, the data structure IPM_PARM_INFO has been filled in. Use SRL functions to retrieve structure fields.

IPMEV_ERROR
indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    ASSUMPTION: A valid nDeviceHandle was obtained from prior
    call to ipm_Open().
    */
    IPM_PARM_INFO ParmInfo;
    unsigned long ulParmValue = 0;
    ParmInfo.eParm = PARMCH_ECHOTAIL;
    ParmInfo.pvParmValue = &ulParmValue;
    if (ipm_GetParm(nDeviceHandle, &ParmInfo, EV_ASYNC)==-1)
    {
        printf("ipm_GetParm failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }

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

```
    }
    else
    {
        printf("%s: ipm_GetParm(parm=0x%x,value=0x%x) ok %\n", ATDV_NAMEP(nDeviceHandle),
            ParmInfo.eParm, ulParmValue );
    }
    /*
    .
    .
    . continue
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_PARM_INFO* pParmInfo;
    switch(nEventType)
    {
        /*
        .
        . Other events
        .
        */
        /* Expected reply to ipm_GetQoSAlarmStatus */
    case IPMEV_GET_PARM:
        pParmInfo = (IPM_PARM_INFO*) pVoid;
        printf("Received IPMEV_GETPARM for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        printf("%s: parm=0x%x, ok %\n", ATDV_NAMEP(nDeviceID),
            pParmInfo->eParm);
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_SetParm( )**

ipm_GetQoSAlarmStatus( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: DM/IP, HMP

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.

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

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

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

Termination Events

IPMEV_GET_QOS_ALARM_STATUS
indicates successful completion, that is, alarm status information was filled in. Use SRL functions to retrieve IPM_QOS_ALARM_STATUS structure fields.

IPMEV_ERROR
indicates 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_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Query the alarm status for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_GetQoSAlarmStatus(nDeviceHandle, NULL, 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()
{
    unsigned int i;
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_QOS_ALARM_STATUS* pAlarmStatus;
    switch(nEventType)
    {
        /*
        .
        . Other events
        .
        */
        /* Expected reply to ipm_GetQoSAlarmStatus */
        case IPMEV_GET_QOS_ALARM_STATUS:
            pAlarmStatus = (IPM_QOS_ALARM_STATUS*)pVoid;
            printf("Received IPMEV_GET_QOS_ALARM_STATUS for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            for(i=0; i < pAlarmStatus->unAlarmCount; ++i)
            {
                switch(pAlarmStatus->QoSData[i].eQoSType)
                {
                    case QOSTYPE_DTMFDISCARDED:
                        printf(" DTMFDISCARDED = %d\n",pAlarmStatus->QoSData[i].eAlarmState);
                        break;
                    case QOSTYPE_LOSTPACKETS:
                        printf(" LOSTPACKETS = %d\n",pAlarmStatus->QoSData[i].eAlarmState);
                        break;
                    case QOSTYPE_JITTER:
                        printf(" JITTER = %d\n",pAlarmStatus->QoSData[i].eAlarmState);
                        break;
                }
            }
            break;
        default:
            printf("Received unknown event = %d for device = %s\n",
                nEventType, ATDV_NAMEP(nDeviceID));
            break;
    }
}
```

See Also

- **ipm_ResetQoSAlarmStatus( )**

ipm_GetQoSThreshold( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

The **ipm_GetQoSThreshold( )** function retrieves QoS alarm threshold settings. Quality of Service (QoS) alarms report the status of a media channel, they do not report alarms for a board.

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

Termination Events

IPMEV_GET_QOS_THRESHOLD_INFO
indicates successful completion, that is, alarm threshold settings were returned. Use SRL functions to retrieve IPM_QOS_THRESHOLD_INFO structure fields.

IPMEV_ERROR
indicates 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_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Query the alarm threshold settings for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    IPM_QOS_THRESHOLD_INFO myThresholdInfo;
    myThresholdInfo.unCount = 0;
    if(ipm_GetQoSThreshold(nDeviceHandle, &myThresholdInfo, EV_ASYNC) == -1)
    {
        printf("ipm_GetQoSAlarmStatus failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
```

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

void CheckEvent()
{
    unsigned int i;
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_QOS_THRESHOLD_INFO* pThresholdInfo;
    switch(nEventType)
    {
        /*
        .
        . Other events
        .
        */
        /* Expected reply to ipm_GetQoSThreshold */
        case IPMEV_GET_QOS_THRESHOLD_INFO:
            pThresholdInfo = (IPM_QOS_THRESHOLD_INFO*)pVoid;
            printf("Received IPMEV_GET_QOS_THRESHOLD_INFO for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            for(i=0; i<pThresholdInfo->unCount; ++i)
            {
                switch(pThresholdInfo->QoSThresholdData[i].eQoSType)
                {
                    case QOSTYPE_DTMFDISCARDED:
                        printf("QOSTYPE_DTMFDISCARDED\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_LOSTPACKETS:
                        printf("QOSTYPE_LOSTPACKETS\n");
                        printf("unTimeInterval = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unTimeInterval);
                        printf("unDebounceOn = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unDebounceOn);
                        printf("unDebounceOff = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unDebounceOff);
                        printf("unFaultThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unFaultThreshold);
                        printf("unPercentSuccessThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unPercentSuccessThreshold);
                        printf("unPercentFailThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unPercentFailThreshold);
                        break;
```

```
                    case QOSTYPE_JITTER:
                        printf("QOSTYPE_JITTER\n");
                        printf("unTimeInterval = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unTimeInterval);
                        printf("unDebounceOn = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unDebounceOn);
                        printf("unDebounceOff = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unDebounceOff);
                        printf("unFaultThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unFaultThreshold);
                        printf("unPercentSuccessThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unPercentSuccessThreshold);
                        printf("unPercentFailThreshold = %d\n",
                            pThresholdInfo->QoSThresholdData[i].unPercentFailThreshold);
                        break;
                    }
                }
                break;
            default:
                printf("Received unknown event = %d for device = %s\n",
                    nEventType, ATDV_NAMEP(nDeviceID));
                break;
        }
    }
```

See Also

- **ipm_SetQoSThreshold( )**

ipm_GetSessionInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: DM/IP, HMP

Description

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

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

A new firmware session is initiated by calling **ipm_StartMedia( )**. In this scenario, data returned by **ipm_GetSessionInfo( )** will be for the current session. **ipm_Stop( )** terminates the session. Between firmware sessions, that is, after **ipm_Stop( )** and before **ipm_StartMedia( )** is called, the data returned by **ipm_GetSessionInfo( )** is for the previous firmware session.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pSessionInfo	pointer to structure that contains Quality of Service (QoS) information about the previous IP session; see IPM_SESSION_INFO for details.
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

- The application can call **ipm_GetQoSAlarmStatus( )** to retrieve alarm information for the current session.
- **ipm_GetSessionInfo( )** is not supported on Intel® NetStructure™ IPT Series boards. If called, it returns zeroes.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Get the current session information for IP device handle, nDeviceHandle.
    ASSUMPTION: nDeviceHandle was obtained from a prior call to ipm_Open().
    Also, ipm_StartMedia() was successfully called
    sometime earlier.
    */
    if(ipm_GetSessionInfo(nDeviceHandle, NULL, EV_ASYNC) == -1)
    {
        printf("ipm_GetSessionInfo failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
```

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

void CheckEvent()
{
    unsigned int i;
    IPM_SESSION_INFO* pIPSessionInfo;
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    void* pVoid = sr_getevtdatap();
    switch(nEventType)
    {
        /*
        .
        .
        . Other events
        .
        .
        */
        /* Expected reply to ipm_GetSessionInfo */
    case IPMEV_GET_SESSION_INFO:
        pIPSessionInfo = (IPM_SESSION_INFO*)pVoid;
        printf("Received IPMEV_GET_SESSION_INFO for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        printf("RtcpInfo.unLocalSR_TimeStamp=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalSR_TimeStamp);
        printf("RtcpInfo.unLocalSR_TxPackets=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalSR_TxPackets);
        printf("RtcpInfo.unLocalSR_TxOctets=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalSR_TxOctets);
        printf("RtcpInfo.unLocalSR_SendIndication=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalSR_SendIndication);
        printf("RtcpInfo.unLocalRR_FractionLost=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalRR_FractionLost);
        printf("RtcpInfo.unLocalRR_CumulativeLost=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalRR_CumulativeLost);
        printf("RtcpInfo.unLocalRR_SeqNumber=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalRR_SeqNumber);
        printf("RtcpInfo.unLocalRR_ValidInfo=%d\n",
            pIPSessionInfo->RtcpInfo.unLocalRR_ValidInfo);
        printf("RtcpInfo.unRemoteSR_TimeStamp=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteSR_TimeStamp);
        printf("RtcpInfo.unRemoteSR_TxPackets=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteSR_TxPackets);
        printf("RtcpInfo.unRemoteSR_TxOctets=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteSR_TxOctets);
        printf("RtcpInfo.unRemoteSR_SendIndication=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteSR_SendIndication);
        printf("RtcpInfo.unRemoteRR_FractionLost=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteRR_FractionLost);
        printf("RtcpInfo.unRemoteRR_CumulativeLost=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteRR_CumulativeLost);
        printf("RtcpInfo.unRemoteRR_SeqNumber=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteRR_SeqNumber);
        printf("RtcpInfo.unRemoteRR_ValidInfo=%d\n",
            pIPSessionInfo->RtcpInfo.unRemoteRR_ValidInfo);
```

```
            for(i = 0; i< pIPSessionInfo->unQoSInfoCount; ++i)
            {
                printf("Session QOS Type=%d\n",
                    pIPSessionInfo->QoSInfo[i].eQoSType);
                printf("Session QOS Data=%d\n",
                    pIPSessionInfo->QoSInfo[i].unData);
            }
            break;
        default:
            printf("Received unknown event = %d for device = %s\n",
                nEventType, ATDV_NAMEP(nDeviceID));
            break;
        }
}
```

See Also

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

ipm_GetXmitSlot( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

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

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

Termination Events

IPMEV_GET_XMITTS_INFO
indicates successful completion, that is, the TDM time slot information data structure was filled in. Use SRL functions to retrieve SC_TSINFO structure fields.

IPMEV_ERROR
indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Get the timeslot information for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_GetXmitSlot(nDeviceHandle, NULL, EV_ASYNC) == -1)
    {
        printf("ipm_GetXmitSlot failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
```

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

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    SC_TSINFO* pTimeSlotInfo;
    switch(nEventType)
    {
    /*
    .
    . Other events
    .
        */
        /* Expected reply to ipm_GetXmitSlot */
    case IPMEV_GET_XMITTS_INFO:
        pTimeSlotInfo = (SC_TSINFO*)pVoid;
        printf("Received IPMEV_GET_XMITTS_INFO for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        printf("Timeslot number %d\n", *(pTimeSlotInfo->sc_tsarrayp));

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

See Also

None.

ipm_Listen( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    SC_TSINFO IPTimeSlotInfo;
    long lTimeSlot;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Tell IP device handle, nDeviceHandle, to listen to timeslot 10.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    lTimeSlot = 10;
    IPTimeSlotInfo.sc_tsarrayp = &lTimeSlot;
    IPTimeSlotInfo.sc_numts = 1;
```

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

void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    switch(nEventType)
    {
    /*
    .
    .
    . Other events
    .
    .
        */
        /* Expected reply to ipm_Listen */
    case IPMEV_LISTEN:
        printf("Received IPMEV_LISTEN for device = %s\n", ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_UnListen( )**

ipm_Open( )

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

Inputs:

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

Returns: device handle if successful
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

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

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

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

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

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

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

Termination Events

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

IPMEV_ERROR
: indicates 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( )**.
- When using Intel® NetStructure™ DM/IP Series boards, you must call **ipm_Open( )** in synchronous mode.
- When using Intel® NetStructure™ Host Media Processing (HMP) software, you must call **ipm_Open( )** in synchronous mode.

Errors

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

EINVAL
: Invalid argument (system-level error)

ENOMEM
: Memory allocation failure (system-level error)

Example

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

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

void main()
{
    char cDevName[10];
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    .
    . Create a Thread that waits on srl events, this
    . thread will execute the WorkerThread function
    .
    */
    /*
```

```
    Open IP channel ipmB1C1
    */
    sprintf(cDevName,"ipmB1C%d", 1);
        if((nDeviceHandle = ipm_Open(cDevName, NULL, EV_ASYNC)) == -1)
        {
            printf("ipm_Open failed for device name = %s\n", cDevName);
            /*
            .
            .
            Perform Error Processing
            .
            .
            */
        }
    /*
    .
    . continue Main Processing
    .
    */
}

void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    switch(nEventType)
    {
    /*
    .
    .
    . Other events
    .
    .
        */
        /* Expected reply to ipm_Open */
    case IPMEV_OPEN:
        printf("Received IPMEV_OPEN for device = %s\n", ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_Close( )**

ipm_Ping( )

Name: int ipm_Ping(nDeviceHandle, *pPingParameter, *pPingInfo, usMode)

Inputs:

int nDeviceHandle	• board device handle
PIPM_PING_PARM *pPingParameter	• pointer to an array of ping parameter structures
IPM_PING_INFO *pPingInfo	• pointer to ping info structure
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: IPT

Description

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

Note: This function is not supported on Intel® NetStructure™ DM/IP Series boards or on the Host Media Processing (HMP) software.

Parameter	Description
nDeviceHandle	handle of the board device **ipmBx**, where **x** is the unique logical board number
***pPingParameter**	pointer to an array of ping parameter structures; see IPM_PING_PARM for details.
pPingInfo	pointer to structure that is filled with ping results upon successful return; see IPM_PING_INFO for details.
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_PING
indicates successful completion, that is, ping response information was returned. Use SRL functions to retrieve IPM_PING_INFO structure fields.

IPMEV_ERROR
indicates the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    ASSUMPTION: A valid nDeviceHandle was obtained from prior
    call to ipm_Open() for a board device.
    */
    IPM_PING_PARM PingParameter;
    strcpy(PingParameter.cRemoteIPAddress, "192.168.1.16");
    strcpy(PingParameter.cLocalIPAddress, "192.168.1.16");
    if(ipm_Ping(nDeviceHandle, &PingParameter, NULL, EV_ASYNC)==-1)
    {
        printf("ipm_Ping failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
    .
```

```
    .
    . continue
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    IPM_PING_INFO* pPingInfo;
    switch(nEventType)
    {
    /*
    .
    . Other events
    .
        */
        /* Expected reply to ipm_GetQoSAlarmStatus */
    case IPMEV_PING:
        pPingInfo = (IPM_PING_INFO*)pVoid;
        printf("Received IPMEV_PING 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_ReceiveDigits( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

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: 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 but that the function was successfully processed by the firmware. The application must enable event reporting and check for the IPMEV_DIGITS_RECEIVED event.

Parameter	Description
nDeviceHandle	handle of the IP Media device
pDigitInfo	pointer to structure that contains digit type, direction, and digits; see IPM_DIGIT_INFO for details. Note that all fields are filled in upon successful function return.
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
indicates 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_DIGIT_INFO struct must have the **unNumberOfDigits** set to 1.
- 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*.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
   int nDeviceHandle;
   IPM_DIGIT_INFO myDigitInfo;

   // Register event handler function with srl
   sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
```

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

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

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

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

void CheckEvent()
{

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

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

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

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

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

See Also

- **ipm_SendDigits( )**

ipm_ResetQoSAlarmStatus( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Platform: DM/IP, HMP

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.

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

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    IPM_QOS_ALARM_STATUS myAlarmStatus;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    */
    /*
    Reset the QOSTYPE_JITTER alarm for IP device handle, nDeviceHandle.
    NOTE: nDeviceHandle was obtained from prior call to ipm_Open()
    */
    myAlarmStatus.unAlarmCount = 1;
    myAlarmStatus.QoSData[0].eQoSType = QOSTYPE_JITTER;
    if(ipm_ResetQoSAlarmStatus(nDeviceHandle, &myAlarmStatus, EV_ASYNC) == -1)
    {
        printf("ipm_ResetQoSAlarmStatus failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
```

```
    .
    . Continue Processing
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
    /*
    .
    . Other events
    .
        */
        /* Expected reply to ipm_ResetQoSAlarmStatus */
    case IPMEV_RESET_QOS_ALARM_STATUS:
        printf("Received IPMEV_RESET_QOS_ALARM_STATUS for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_GetQoSAlarmStatus( )**

ipm_SendDigits( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Platform: DM/IP, IPT

Description

The **ipm_SendDigits( )** function generates the supplied digits in the specified direction.

Note: This function is not supported on Intel® NetStructure™ Host Media Processing (HMP) software.

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

- If this function is called synchronously and returns 0, it does not indicate that the digits have been sent, but 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.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    IPM_DIGIT_INFO myDigitInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Generate a set of digits using IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    myDigitInfo.eDigitType = DIGIT_ALPHA_NUMERIC;
    myDigitInfo.eDigitDirection = DIGIT_TDM;
    strcpy(myDigitInfo.cDigits,"12345678901234567890");
    myDigitInfo.unNumberOfDigits = 20;
    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

Platform: DM/IP, HMP

Description

The **ipm_SendRFC2833SignalIDToIP( )** function sends the supplied RFC 2833 signal to IP.

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pSignalInfo	pointer to structure that contains RFC 2833 signal ID and state information; see IPM_RFC2833_SIGNALID_INFO for details. Note that the application must fill in the RFC 2833 signal to be sent.
usMode	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 the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <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_SendDigits()
    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: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

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

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    ASSUMPTION: A valid nDeviceHandle was obtained from prior
    call to ipm_Open().
    */
    IPM_PARM_INFO ParmInfo;
    unsigned long ulParmValue = ECHO_TAIL_16;
    ParmInfo.eParm = PARMCH_ECHOTAIL;
    ParmInfo.pvParmValue = &ulParmValue;
    if(ipm_SetParm(nDeviceHandle, &ParmInfo, EV_ASYNC)==-1)
    {
        printf("ipm_SetParm failed for device name %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
    .
    .
    . continue
    .
    */
}
```

```
void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    void* pVoid = sr_getevtdatap();
    switch(nEventType)
    {
    /*
    .
    . Other events
    .
        */
        /* Expected reply to ipm_GetQoSAlarmStatus */
    case IPMEV_SET_PARM:
        printf("Received IPMEV_SETPARM for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_GetParm( )**

ipm_SetQoSThreshold( )

Name: int ipm_SetQoSThreshold(nDeviceHandle, *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

Platform: DM/IP, IPT, HMP

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

Termination Events

IPMEV_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 the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    IPM_QOS_THRESHOLD_INFO mySetQosThresholdInfo;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Change two alarm threshold settings for IP device handle, nDeviceHandle.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    mySetQosThresholdInfo.unCount = 2;
    mySetQosThresholdInfo.QoSThresholdData[0].eQoSType = QOSTYPE_LOSTPACKETS;
    mySetQosThresholdInfo.QoSThresholdData[0].unTimeInterval = 10;
    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 = 50;
    mySetQosThresholdInfo.QoSThresholdData[1].unDebounceOn = 200;
```

```
    mySetQosThresholdInfo.QoSThresholdData[1].unDebounceOff = 600;
    mySetQosThresholdInfo.QoSThresholdData[1].unFaultThreshold = 60;
    mySetQosThresholdInfo.QoSThresholdData[1].unPercentSuccessThreshold = 60;
    mySetQosThresholdInfo.QoSThresholdData[1].unPercentFailThreshold = 40;
    if(ipm_SetQoSThreshold(nDeviceHandle, &mySetQosThresholdInfo, EV_ASYNC) == -1)
    {
        printf("ipm_SetQoSThreshold failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
    .
    .
    . continue
    .
    */
}

void CheckEvent()
{
    //Get event type and associated data
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
    /*
    .
    . Other events
    .
        */
        /* Expected reply to ipm_SetQoSThreshold */
    case IPMEV_SET_QOS_THRESHOLD_INFO:
        printf("Received IPMEV_SET_QOS_THRESHOLD_INFO for device = %s\n",
            ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_GetQoSThreshold( )**

ipm_SetRemoteMediaInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

Note: We strongly recommend that you use the **ipm_StartMedia( )** function instead of **ipm_SetRemoteMediaInfo( )**. Support for the **ipm_SetRemoteMediaInfo( )** function may be removed from future versions of the IP media API.

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
nDeviceHandle	handle of the IP Media device
pMediaInfo	pointer to structure; see IPM_MEDIA_INFO for details. Applications can define the following: • local transmit coder and remote transmit coder • local and remote RTP/RTCP protocol • local and remote IP address

Parameter	Description
eDirection	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: • DATA_IP_RECEIVEONLY – receives data from the IP network but no data is sent. • DATA_IP_SENDONLY – sends data to the IP network but no data is received. • DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM. Used for gateway functionality. • DATA_MULTICAST_SERVER – multicast server mode • DATA_MULTICAST_CLIENT – multicast client mode
usMode	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 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_BADPARM
Invalid parameter

EIPM_BUSY
Channel is busy

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <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_FRAMESIZE) 30;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unFramesPerPkt = 1;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unCoderPayloadType = 0;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unRedPayloadType = 0;
    MediaInfo.MediaData[3].eMediaType = MEDIATYPE_LOCAL_CODER_INFO;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unFramesPerPkt = 1;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eVadEnable =CODER_VAD_DISABLE;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unCoderPayloadType = 0;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unRedPayloadType = 0;
    if(ipm_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

Platform: DM/IP, IPT, HMP

Description

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pMediaInfo	pointer to structure; see IPM_MEDIA_INFO for details. Applications can define the following: • local transmit coder and remote transmit coder • local and remote RTP/RTCP protocol • local and remote IP address

Parameter	Description
eDirection	media operation enumeration The eIPM_DATA_DIRECTION data type is an enumeration which defines the following values: • DATA_IP_RECEIVEONLY – receives data from the IP network but no data is sent. • DATA_IP_SENDONLY – sends data to the IP network but no data is received. • DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM. Used for gateway functionality. • DATA_MULTICAST_SERVER – multicast server mode • DATA_MULTICAST_CLIENT – multicast client mode
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_START_MEDIA
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 the function failed.

Cautions

The application must wait until this function completes before calling **ipm_Listen( )**.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_BUSY
Channel is busy

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

```
#include <stdio.h>
#include <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_FRAMESIZE) 30;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unFramesPerPkt = 1;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.eVadEnable = CODER_VAD_DISABLE;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unCoderPayloadType = 0;
    MediaInfo.MediaData[2].mediaInfo.CoderInfo.unRedPayloadType = 0;
    MediaInfo.MediaData[3].eMediaType = MEDIATYPE_LOCAL_CODER_INFO;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_FRAMESIZE) 30;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unFramesPerPkt = 1;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.eVadEnable =CODER_VAD_DISABLE;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unCoderPayloadType = 0;
    MediaInfo.MediaData[3].mediaInfo.CoderInfo.unRedPayloadType = 0;
    if(ipm_StartMedia(nDeviceHandle, &MediaInfo,DATA_IP_TDM_BIDIRECTIONAL, EV_ASYNC) == -1)
    {
        printf("ipm_StartMediaInfo failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
    .
    . Continue processing
    .
    */
}
```

```
void CheckEvent()
{
    int nDeviceID = sr_getevtdev();
    int nEventType = sr_getevttype();
    switch(nEventType)
    {
    /*
    .
    .
    . Other events
    .
    .
        */
        /* Expected reply to ipm_StartMedia */
    case IPMEV_STARTMEDIA:
        printf("Received IPMEV_START_MEDIA 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_Stop( )**

ipm_Stop( )

Name: int ipm_Stop(nDeviceHandle, eOperation, usMode)

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

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

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
eOperation	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: • STOP_SEND_DIGITS – operation of sending digits • STOP_RECEIVE_DIGITS – operation of receiving digits • STOP_RECEIVE_DIGITS_RFC2833 – operation of receiving RFC 2833 digits • STOP_MEDIA – operation of media session.This enumeration disconnects the session. The application must call **ipm_StartMedia( )** to start a new session. • STOP_ALL – stop all operations
usMode	operation mode. Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

IPMEV_STOPPED
: indicates that activity of the type specified in eIPM_STOP_OPERATION has terminated on this channel. This event does not return data.

IPMEV_ERROR
: indicates the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
: Invalid parameter

EIPM_FWERROR
: Firmware error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    . Main Processing
    .
    .
    */
    /*
    Application needs to stop a current session on IP device handle, nDeviceHandle
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open()
    and a session has been started by calling ipm_StartMedia()
    sometime earlier.
    */
    if(ipm_Stop(nDeviceHandle, STOP_ALL, EV_ASYNC) == -1)
    {
        printf("ipm_Stop failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
```

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

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
    /*
    .
    . List of expected events
    .
        */
        /* Expected reply from ipm_Stop() */
    case IPMEV_STOPPED:
        printf("Received IPMEV_STOPPED for device = %s\n", ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_UnListen( )**

ipm_UnListen( )

Name: int ipm_UnListen(nDeviceHandle, usMode)

Inputs:

int nDeviceHandle	• IP Media device handle
unsigned short usMode	• async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Platform: DM/IP, IPT, HMP

Description

The **ipm_UnListen( )** function stops listening to the TDM time slot specified in a previous call to **ipm_Listen( )**. When **ipm_Stop( )** is called to stop a media session on DM3 hardware, **ipm_UnListen( )** is called automatically.

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

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

Termination Events

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

IPMEV_ERROR
indicates the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

void main()
{
    int nDeviceHandle;
    // Register event handler function with srl
    sr_enbhdlr( EV_ANYDEV ,EV_ANYEVT ,(HDLR)CheckEvent);
    /*
    .
    .
    Main Processing
    .
    .
    .
    */
    /*
    Stop an IP device handle, nDeviceHandle, from listening to a time slot.
    ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if(ipm_UnListen(nDeviceHandle, EV_ASYNC) == -1)
    {
        printf("ipm_UnListen failed for device name = %s with error = %d\n",
            ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
            /*
            .
            .
            Perform Error Processing
            .
            .
        */
    }
    /*
```

```
    .
    . Continue processing
    .
    */
}

void CheckEvent()
{
    int nEventType = sr_getevttype();
    int nDeviceID = sr_getevtdev();
    switch(nEventType)
    {
    /*
    .
    .
    . Other events
    .
    .
        */
        /*Expected reply from ipm_UnListen*/
    case IPMEV_UNLISTEN:
        printf("Received IPMEV_UNLISTEN for device = %s\n", ATDV_NAMEP(nDeviceID));
        break;
    default:
        printf("Received unknown event = %d for device = %s\n",
            nEventType, ATDV_NAMEP(nDeviceID));
        break;
    }
}
```

See Also

- **ipm_Listen( )**
- **ipm_Stop( )**

Events 3

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

There are three types of events returned by the IP Media software functions:

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

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

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

IPMEV_DIGITS_RECEIVED
: Unsolicited event for **ipm_ReceiveDigits( )** in either synchronous or asynchronous mode. 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_PING
Termination event for **ipm_Ping( )**. IPM_PING_INFO contains data. Indicates ping response has been returned.

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_SET_REMOTE_MEDIA_INFO
Termination event for **ipm_SetRemoteMediaInfo( )**. IPM_MEDIA_INFO contains data. Indicates media channel information has been set and session has been started.

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.

Data Structures 4

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.

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_FRAMESIZE    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. The structure is used by the **ipm_GetLocalMediaInfo( )** and **ipm_SetRemoteMediaInfo( )** functions.

Appropriate values for IPM_CODER_INFO fields depend on the board that is being used. Table 2 and Table 3 list supported coders for Intel® NetStructure™ IPT Series boards and Intel® NetStructure™ DM/IP Series boards.

Intel® NetStructure™ Host Media Processing (HMP) software performs voice, conferencing and IVR processing on general-purpose servers based on Intel® architecture without the use of specialized hardware. Table 4 shows the coders that are supported when using the IP media API with HMP.

Field Descriptions

The fields of the IPM_CODER_INFO data structure are described as follows. Refer to Table 2, Table 3, and Table 4 for platform-specific guidelines for filling in these fields.

eCoderType

type of coder to be used for streaming media operations. Coder-specific values for this field are listed in Table 2, Table 3, and Table 4.

The following values are supported:

- 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_G726_32K – G.726.3, 32 kbps
- CODER_TYPE_G729 – G.729
- CODER_TYPE_G729ANNEXA – G.729 Annex A
- CODER_TYPE_G729ANNEXB – G.729 Annex B
- CODER_TYPE_G729ANNEXAWANNEXB – G.729 Annex A with Annex B
- CODER_TYPE_GSMFULLRATE – GSM (TIPHON), full rate (Intel® NetStructure™ DM/IP Series boards only)

eFrameSize

size of frame (G.711 coders only). When packets are sent in both directions, (that is, when the call to **ipm_StartMedia( )** or **ipm_SetRemoteMediaInfo( )** specifies

eDirection = DATA_IP_TDM_BIDIRECTIONAL), the application must know the frame size of incoming packets and use eIPM_CODER_FRAMESIZE to specify that value.

The eIPM_CODER_FRAMESIZE data type is an enumeration which specifies the frame size for G.711 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. The following values for eIPM_CODER_FRAMESIZE are supported:

- CODER_FRAMESIZE_5 – frame size = 5 ms (Intel® NetStructure™ IPT Series boards only)
- CODER_FRAMESIZE_10 – frame size = 10 ms
- CODER_FRAMESIZE_20 – frame size = 20 ms
- CODER_FRAMESIZE_30 – frame size = 30 ms

unFramesPerPkt

number of frames per packet. Coder-specific values for this field are listed in Table 2, Table 3, and Table 4. 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: 0-127. 96-127 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. Value: 96-127

Table 2. Supported Coders for Intel® NetStructure™ IPT Series Boards

Coder	Frame Size (ms)	Frames per Packet (fpp)	VAD Support
CODER_TYPE_G711ALAW64K	5, 10, 20, 30	fixed at 1	N/A
CODER_TYPE_G711ULAW64K	5, 10, 20, 30	fixed at 1	N/A
CODER_TYPE_G7231_5_3K	fixed at 30	1, 2, 3, 4	Supported
CODER_TYPE_G7231_6_3K	fixed at 30	1, 2, 3, 4	Supported
CODER_TYPE_G726_32K (see Note)	10	1, 2, or 3	Supported
	20	1 or 2 (transmit) 1, 2, or 3 (receive)	
	30	1 (transmit) 1 or 2 (receive)	

NOTE: G.726 coders have the following limitations:
(Frames per Packet) x (Frame size) cannot be > 40 for the transmit (remote) side
(Frames per Packet) x (Frame size) cannot be > 60 for the receive (local) side

Table 2. Supported Coders for Intel® NetStructure™ IPT Series Boards (Continued)

Coder	Frame Size (ms)	Frames per Packet (fpp)	VAD Support
CODER_TYPE_G729	fixed at 30	1, 2, 3, or 4	N/A
CODER_TYPE_G729ANNEXA	fixed at 30	1, 2, 3, or 4	N/A
CODER_TYPE_G729ANNEXB	fixed at 30	1, 2, 3, or 4	Supported
CODER_TYPE_G729ANNEXAWANNEXB	fixed at 30	1, 2, 3, or 4	Supported
NOTE: G.726 coders have the following limitations: (Frames per Packet) x (Frame size) cannot be > 40 for the transmit (remote) side (Frames per Packet) x (Frame size) cannot be > 60 for the receive (local) side			

Table 3. Supported Coders for Intel® NetStructure™ DM/IP Series Boards

Coder	Frame Size (ms)	Frames per Packet (fpp)	VAD Support
CODER_TYPE_G711ALAW64K	10, 20, or 30	fixed at 1	N/A
CODER_TYPE_G711ULAW64K	10, 20, or 30	fixed at 1	N/A
CODER_TYPE_G7231_5_3K	fixed at 30	1, 2, or 3	Supported
CODER_TYPE_G7231_6_3K	fixed at 30	1, 2, or 3	Supported
CODER_TYPE_G726_32K[1]	N/A	N/A	N/A
CODER_TYPE_G729	fixed at 10	1, 2, 3, or 4	N/A
CODER_TYPE_G729ANNEXA	fixed at 10	1, 2, 3, or 4	N/A
CODER_TYPE_G729ANNEXB	fixed at 10	1, 2, 3, or 4	N/A
CODER_TYPE_G729ANNEXAWANNEXB	fixed at 10	1, 2, 3, or 4	N/A
CODER_TYPE_GSMFULLRATE[2]	fixed at 20	1, 2, or 3	Supported
NOTES: 1. G.726 support is limited to play and record functionality only; transcoding is not supported on this coder. 2. GSM Telecommunications and Internet Protocol Harmonization over Networks (TIPHON) is a sub-group of the European Telecommunications Standards Institute (ETSI) GSM specification.			

Table 4. Supported Coders for Host Media Processing

Coder	Frame Size (ms)	Frames per Packet (fpp)	VAD Support
CODER_TYPE_G711ALAW64K	10, 20, or 30	fixed at 1	N/A
CODER_TYPE_G711ULAW64K	10, 20, or 30	fixed at 1	N/A

IPM_DIGIT_INFO

```
typedef struct ipm_digit_info_tag
{
    eIPM_DIGIT_TYPE eDigitType;             /* Type of digits - DTMF, ALPHA-NUMERIC */
    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]
actual digits to be sent or received; maximum number of digits = 32

unNumberOfDigits
number of digits; must be set to 1.

unTimeStamp
set to 0; reserved for future use

unExpirationTime
set to 0; reserved for future use

unDuration
set to 0; reserved for future use

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_SetRemoteMediaInfo( )** functions.

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_SetRemoteMediaInfo( )** and **ipm_GetLocalMediaInfo( )** functions.

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_REMOTE_RTP_INFO – remote RTP port information
- MEDIATYPE_LOCAL_RTP_INFO – local RTP port information
- MEDIATYPE_REMOTE_RTCP_INFO – remote RTCP port information
- MEDIATYPE_LOCAL_RTCP_INFO – local RTCP port information
- MEDIATYPE_REMOTE_CODER_INFO – remote receive coder information
- MEDIATYPE_LOCAL_CODER_INFO – local receive coder information
- MEDIATYPE_FAX_SIGNAL_INFO – fax signal information to be transmitted towards IP during fax transmissions
- MEDIATYPE_LOCAL_UDPTL_T38_INFO – local UDP packet T.38 information
- MEDIATYPE_REMOTE_UDPTL_T38_INFO – remote UDP packet T.38 information

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 **ipm_SetRemoteMediaInfo( )** and **ipm_GetLocalMediaInfo( )**.

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 5 for values.

*pvParmValue

pointer to the value of the parameter

Table 5. eIPM_PARM Values

Define	Description
PARMCH_ECHOTAIL	echo tail length value. Supported values for Intel® NetStructure™ DM/IP Series boards include: ECHO_TAIL_NONE, ECHO_TAIL_8, ECHO_TAIL_16, ECHO_TAIL_32 Supported values for Intel® NetStructure™ IPT Series boards include: ECHO_TAIL_NONE, ECHO_TAIL_8, ECHO_TAIL_16, ECHO_TAIL_32, ECHO_TAIL_48, ECHO_TAIL_64, ECHO_TAIL_96, ECHO_TAIL_128
PARMCH_RFC2833REDLEVEL	redundancy level; (supported on Intel® NetStructure™ DM/IP Series boards only) values include: RFC2833REDLEVEL_1, RFC2833REDLEVEL_2, RFC2833REDLEVEL_3, RFC2833REDLEVEL_4, RFC2833REDLEVEL_5
PARMCH_RFC2833GEN_TO_TDM	convert RFC2833 to signal (Intel® NetStructure™ DM/IP Series boards only) Values are: RFC2833GEN_TO_TDM_OFF, RFC2833GEN_TO_TDM_ON
PARMCH_RFC2833GEN_TO_IP	send RFC2833 to IP (OFF / ON) (Intel® NetStructure™ DM/IP Series boards only) Values are: RFC2833GEN_TO_IP_OFF, RFC2833GEN_TO_IP_ON

Table 5. eIPM_PARM Values (Continued)

Define	Description
PARMCH_DTMFXFERMODE	DTMF transfer mode; values include: DTMFXFERMODE_INBAND in-band (default) DTMFXFERMODE_OUTOFBAND out-of-band DTMFXFERMODE_RFC2833 RFC 2833 **Note:** In order for DTMF event reporting to occur, you must set eIPM_DTMFXFERMODE to out-of-band signaling on the receive side.
PARMCH_ECACTIVE	echo cancellation active. Values are: ECACTIVE_OFF, ECACTIVE_ON
PARMCH_AGCACTIVE	automatic gain control active (Intel® NetStructure™ DM/IP Series boards only). Values are: AGCACTIVE_OFF, AGCACTIVE_ON
PARMCH_TOS	type of service, range = 0-255
PARMCH_RFC2833EVT_TX_PLT	RFC2833 event transmit payload
PARMCH_RFC2833EVT_RX_PLT	RFC2833 event receive payload
PARMCH_RFC2833TONE_TX_PLT	RFC2833 tone transmit payload
PARMCH_RFC2833TONE_RX_PLT	RFC2833 tone receive payload
PARMCH_RFC2833MUTE_AUDIO	clamping mode for DTMF data sent using RFC 2833 mode. This parameter is set to ON by default. Values are: RFC2833MUTE_AUDIO_ON (default), RFC2833MUTE_AUDIO_OFF **Note:** This parameter is invalid for in-band mode, since the data is automatically transmitted on the RTP stream. This parameter is not applicable to out-of-band mode, since the data is automatically muted (clamped). See the *IP Media Library API Programming Guide* for more details.

IPM_PING_INFO

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

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

Description

This structure contains ping response information. The structure is used by the **ipm_Ping( )** function.

Field Descriptions

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

unPacketsSent
: number of packets sent

unPacketsReceived
: number of packets received

unPacketsLost
: number of packets lost

fRoundTripMin
: minimum round trip time in msec

fRoundTripAvg
: average round trip time in msec

fRoundTripMax
: maximum round trip time in msec

IPM_PING_PARM

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

Description

This structure contains ping parameter information. The structure is used by the **ipm_Ping( )** function.

Note: For a board device, the value for cLocalIPAddress can be obtained by calling **ipm_GetParm( )**. For a channel device, **ipm_GetLocalMediaInfo( )** should be used. However, the IP addresses returned from **ipm_GetParm( )** will work for channel devices.

Field Descriptions

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

cRemoteIPAddress[IP_ADDR_SIZE]
destination IP address; null-terminated string formatted as standard dotted-decimal IP address

cLocalIPAddress[IP_ADDR_SIZE]
local board IP address; null-terminated string formatted as standard dotted-decimal IP address

ulNumOfPings
reserved for future use (RFU)

ulPacketSize
reserved for future use (RFU)

ulTimeout
reserved for future use (RFU)

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 and RTCP 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]
: IP address of the port in standard dotted decimal string format; must be null-terminated. For example, 192.168.0.1

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( )**.

Field Descriptions

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

eQoSType
identifies the QoS alarm that is to be set or reset

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

- EVT_DTMFDISCARDED – number of lost DTMF digits since the beginning of the call (Intel® NetStructure™ DM/IP Series boards only)
- EVT_LOSTPACKETS – percent of lost packets since the beginning of the call
- EVT_JITTER – average jitter since the beginning of the call (in msec)
- EVT_ROUNDTRIPLATENCY – RTP packet latency (Intel® NetStructure™ IPT Series boards only)

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

Notes: ***1.*** For Intel® NetStructure™ IPT Series boards, the system software sends a QoS alarm event when a threshold is exceeded (ALARM_STATE_ON).

2. For Intel® NetStructure™ DM/IP Series boards, the system software sends a QoS alarm event when a threshold is exceeded (ALARM_STATE_ON) and when the threshold returns to the programmed level (ALARM_STATE_OFF).

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

Note: EVT_DTMFDISCARDED is not supported on Intel® NetStructure™ IPT Series boards.
EVT_ROUNDTRIPLATENCY is not supported on Intel® NetStructure™ DM/IP Series boards.

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

- EVT_DTMFDISCARDED – number of lost DTMF digits since the beginning of the call
- EVT_LOSTPACKETS – percent of lost packets since the beginning of the call
- EVT_JITTER – average jitter since the beginning of the call (in msec)
- EVT_ROUNDTRIPLATENCY – RTP packet latency

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( )**.

Field Descriptions

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

eQoSType
QoS parameter type

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

- EVT_DTMFDISCARDED – number of lost DTMF digits since the beginning of the call (Intel® NetStructure™ DM/IP Series boards only)
- EVT_LOSTPACKETS – percent of lost packets since the beginning of the call
- EVT_JITTER – average jitter since the beginning of the call (in msec)
- EVT_ROUNDTRIPLATENCY – RTP packet latency (Intel® NetStructure™ IPT Series boards only)

unTimeInterval
time interval (in 100 ms units)

Note: This field is not supported on Intel® NetStructure™ IPT Series boards.

unDebounceOn
debounce on time (in 100 ms units); multiple of unTimeInterval

Note: This field is not supported on Intel® NetStructure™ IPT Series boards.

unDebounceOff
debounce off time (in 100 ms units); multiple of unTimeInterval

Note: This field is not supported on Intel® NetStructure™ IPT Series boards.

unFaultThreshold
fault threshold parameter

unPercentSuccessThreshold
threshold of successes during unDebounceOff time (expressed as a percentage of successes)

Note: This field is not supported on Intel® NetStructure™ IPT Series boards.

unPercentFailThreshold

threshold of failures during unDebounceOn time (expressed as a percentage of failures)

Note: This field is not supported on Intel® NetStructure™ IPT Series boards.

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.

Note: This structure is not supported on Intel® NetStructure™ IPT Series boards.

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 listed in Table 6 and Table 7 for Intel® NetStructure™ DM/IP Series Boards and Host Media Processing (HMP) software respectively.

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, and tone is sent. ie, HOLDING down the key until turn off state. no defulat must be set.

Table 6. eIPM_RFC2833_SIGNAL_ID Values for DM/IP Series Boards

Name	Value (H)
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

Table 6. eIPM_RFC2833_SIGNAL_ID Values for DM/IP Series Boards (Continued)

Name	Value (H)
SIGNAL_ID_EVENT_DTMF_A	0xc
SIGNAL_ID_EVENT_DTMF_B	0xd
SIGNAL_ID_EVENT_DTMF_C	0xe
SIGNAL_ID_EVENT_DTMF_D	0xf
SIGNAL_ID_EVENT_LINE_RINGING_TONE	0x46

Table 7. eIPM_RFC2833_SIGNAL_ID Values for HMP Software

Name	Value (H)
SIGNAL_ID_EVENT_DTMF_1	0x1
SIGNAL_ID_EVENT_DTMF_2	0x2
SIGNAL_ID_EVENT_DTMF_3	0x3
SIGNAL_ID_EVENT_DTMF_4	0x4
SIGNAL_ID_EVENT_DTMF_5	0x5
SIGNAL_ID_EVENT_DTMF_6	0x6
SIGNAL_ID_EVENT_DTMF_7	0x7
SIGNAL_ID_EVENT_DTMF_8	0x8
SIGNAL_ID_EVENT_DTMF_9	0x9
SIGNAL_ID_EVENT_DTMF_STAR	0xa
SIGNAL_ID_EVENT_DTMF_POUND	0xb
SIGNAL_ID_EVENT_DTMF_A	0xc
SIGNAL_ID_EVENT_DTMF_B	0xd
SIGNAL_ID_EVENT_DTMF_C	0xe
SIGNAL_ID_EVENT_DTMF_D	0xf

IPM_RTCP_SESSION_INFO

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

} IPM_RTCP_SESSION_INFO, *PIPM_RTCP_SESSION_INFO;
```

Description

This structure contains RTCP information for the session. It is a child of the IPM_SESSION_INFO structure which is filled in when **ipm_GetSessionInfo( )** returns successfully.

Note: The structure is not supported for Intel® NetStructure™ IPT Series boards.

Field Descriptions

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

unLocalSR_TimeStamp
time stamp of the RTCP packet transmission from the local sender

unLocalSR_TxPackets
number of packets sent by the local sender

unLocalSR_TxOctets
number of bytes sent by the local sender

unLocalSR_SendIndication
local sender report has changed since the last transmission. Values may be either:

- FALSE
- TRUE

unLocalRR_FractionLost
percentage of packets lost, as computed by the local receiver

unLocalRR_CumulativeLost
number of packets lost, as computed by the local receiver

unLocalRR_SeqNumber
last sequence number received from the local receiver

unLocalRR_ValidInfo
reserved for future use

unRemoteSR_TimeStamp
time stamp of the RTCP packet transmission from the remote sender

unRemoteSR_TxPackets
number of packets sent by the remote sender

unRemoteSR_TxOctets
number of bytes sent by the remote sender

unRemoteSR_SendIndication
remote sender report has changed since the last transmission. Values may be either:

- FALSE
- TRUE

unRemoteRR_FractionLost
percentage of packets lost, as computed by the remote receiver

unRemoteRR_CumulativeLost
number of packets lost, as computed by the remote receiver

unRemoteRR_SeqNumber
last sequence number received from the remote receiver

unRemoteRR_ValidInfo
reserved for future use

IPM_SESSION_INFO

```
typedef struct ipm_session_info_tag
{
    IPM_RTCP_SESSION_INFO  RtcpInfo;
    unsigned int           unQoSInfoCount;
    IPM_QOS_SESSION_INFO   QoSInfo[MAX_QOS_SESSION];

} IPM_SESSION_INFO, *PIPM_SESSION_INFO;
```

Description

This structure is a parent structure of the IPM_RTCP_SESSION_INFO and IPM_QOS_SESSION_INFO structures, and it is used by the **ipm_GetSessionInfo( )** function. It reports QoS statistics during the last IP session, including RTCP information. Note that it does not contain statistics for the current IP session.

Note: This structure is not supported on Intel® NetStructure™ IPT Series boards.

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
: must be set to 1 for this release; number of time slots to follow.

sc_tsarrayp
: time slot ID number

Error Codes 5

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_BADPARM
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_Ping( )**, **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_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( )**

Index

T