IP Media Library API for Host Media Processing

Library Reference

August 2005

05-2257-005

Publication Date: August 2005

Document Number: 05-2257-005

Intel Converged Communications, Inc.
1515 Route 10
Parsippany, NJ 07054

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

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

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

Contents

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2257-005	August 2005	ipm_GetLocalMediaInfo( ) function: Updated code example for video media info IPM_AUDIO_CODER_INFO data a structure: Added note about name change; added note to unCoderPayloadType description (PTR#33921) IPM_MEDIA data structure: Added union element for video coders; changed name of element for audio coders IPM_PARM_INFO data structure: Added info on using PARMCH_TOS for DSCP field IPM_RFC2833_SIGNALID_INFO data structure: Restored to document (previous removal was in error) IPM_VIDEO_CODER_INFO: New data structure
05-2257-004	April 2005	Function Summary by Category chapter: Removed unimplemented ipm_SendRFC2388SignalIDToIP function ipm_DisableEvents( ) function page: Removed unsupported EVT_FAXTONE and EVT_T38CALLSTATE event types ipm_EnableEvents( ) function page: Removed unsupported EVT_FAXTONE and EVT_T38CALLSTATE event types ipm_SendRFC2388SignalIDToIP function: Removed as unimplemented ipm_SetQoSThreshold( ) function page: Added Caution on possible failure scenario. Corrected code example Events chapter: Removed unimplemented IPMEV_RFC2833SIGNALRECEIVED event IPM_RFC2833_SIGNALID_INFO data structure section: Removed as unused IPM_QOS_SESSION_INFO data structure page: Clarified descriptions of jitter and lost packets QoS statistics
05-2257-003	September 2004	ipm_GetQoSAlarmStatus( ) function: Corrected code example ipm_GetSessionInfo( ) function: Added info on NULL pointer in asynch mode ipm_GetXmitSlot( ) function: Added info on NULL pointer in asynch mode ipm_Open( ) function: Removed caution that function must be called in synchronous mode ipm_ReceiveDigits( ) function: Clarified usage of data structure. Added caution regarding active RTP session requirement for receiving digits. ipm_SendDigits( ) function: Added info about maximum number of digits. Added caution regarding active RTP session requirement for sending digits. ipm_StartMedia( ) function: Added caution to avoid setting IP address 0.0.0.0 (PTR#32986). Corrected name of completion event. CT_DEVINFO data structure: Removed much information irrelevant to the structure's use with IPML on HMP

Document No.	Publication Date	Description of Revisions
05-2257-003 (continued)		IPM_DIGIT_INFO data structure: Added info about maximum number of digits and send vs. receive usage differences IPM_PARM_INFO data structure: Removed descriptions of three unsupported RFC2833 parameters IPM_PORT_INFO data structure: Added caution to avoid setting IP address 0.0.0.0 (PTR#32986)
05-2257-002	April 2004	Made document HMP-specific by removing hardware-specific information, including "IP Media Function Support by Platform" section. ipm_DisableEvents( ) function: Removed two unsupported events. Added EVT_RTCPTIMEOUT and EVT_RTPTIMEOUT QoS alarm events. ipm_EnableEvents( ) function: Removed two unsupported events. Added EVT_RTCPTIMEOUT and EVT_RTPTIMEOUT QoS alarm events. ipm_GetLocalMediaInfo( ) function: Corrected MEDIATYPE... types referred to in the Cautions and Code Example sections ipm_Ping( ): Removed as unsupported ipm_SendDigits( ) function: Documented as supported for HMP ipm_SetQoSThreshold( ) function: Revised code example to use correct minimum value (100) for unTimeInterval ipm_SetRemoteMediaInfo( ) function: Documented function as deprecated. Corrected code example. ipm_StartMedia( ) function: Corrected code example ipm_Stop( ) function: Removed two unsupported eIPM_STOP_OPERATION values IPM_CODER_INFO data structure: Removed hardware-specific enum values and tables of supported coders. Updated table of supported coders. IPM_PARM_INFO data structure: Added PARMCH_RX_ADJVOLUME and PARMCH_RX_ADJVOLUME parameters. Expanded and corrected descriptions of all parameters. Removed hardware-specific enum values. IPM_PING_INFO and IPM_PINGPARM structures: Removed as unsupported IPM_QOS_ALARM_DATA data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms. IPM_QOS_SESSION_INFO data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms. IPM_QOS_THRESHOLD_DATA data structure: Corrected names of eIPM_QOS_TYPE enums. Added new enums for RTCP Timeout and RTP Timeout alarms. Added value ranges for unFaultThreshold. Updated descriptions of all fields. Added table of default parameter values. Error Codes chapter: added EIPM_RESOURCEINUSE
05-2257-001	September 2003	Initial version of HMP-specific document. Much of the information contained in this document was previously published in the *IP Media Library API for Linux and Windows Operating Systems Library Reference*, document number 05-1833-002. Among other small changes, the following changes were made: ipm_GetCTInfo( ): New function CT_DEVINFO: Added to book IPM_PARM_INFO data structure: Added valid value and variable type info for PARMCH_RFC2833EVT_TX_PLT and PARMCH_RFC2833EVT_RX_PLT. Deleted PARMCH_RFC2833MUTE_AUDIO, PARMCH_RFC2833TONE_TX_PLT, and PARMCH_RFC2833TONE_RX_PLT

About This Publication

The following topics provide information about this publication:

Purpose

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

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

Intended Audience

This guide is intended for software developers who will access the IP media software. This may include any of the following:

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

How to Use This Publication

Refer to this publication after you have installed the hardware and the system software which includes the IP media software. This publication assumes that you are familiar with the Linux* or Windows* operating system and the C programming language. It is helpful to keep the *Voice API Library Reference* handy as you develop your application.

The information in this guide is organized as follows:

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

Related Information

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

- *IP Media Library API Programming Guide*
- *Global Call IP Technology Guide*
- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *Standard Runtime Library API for Linux and Windows Operating Systems Library Reference*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/design/network/products/telecom/* (for product information)

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_SetParm( )
sets IP channel parameters

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

1.2 I/O (Input/Output) Functions

The following functions are used to transfer digits and data:

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

ipm_SendDigits( )
generates supplied digits to the TDM bus

1.3 Media Session Functions

The following functions are used to perform session management:

ipm_GetCTInfo( )
retrieves information about an IPM device voice channel

ipm_GetLocalMediaInfo( )
retrieves properties for the local media channel

ipm_GetSessionInfo( )
retrieves statistics for the current session

ipm_SetRemoteMediaInfo( )
This function is deprecated and included only for backwards compatibility; use **ipm_StartMedia( )** instead.

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

ipm_Stop( )
stops operations on an IP channel

1.4 Quality of Service (QoS) Functions

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

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

ipm_GetQoSThreshold( )
retrieves QoS alarm threshold settings

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

ipm_SetQoSThreshold( )
changes QoS alarm threshold settings

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

Description

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

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

Termination Events

None - this function operates in synchronous mode only.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_CONFIG
Configuration error

EIPM_FWERROR
Firmware error

Example

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

void main()
{
    int nDeviceHandle;

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

See Also

- **ipm_Open( )**

ipm_DisableEvents( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pEvents	array of enumerations that specifies the events to disable The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_LOSTPACKETS – QoS alarm for excessive percentage of lost packets • EVT_JITTER – QoS alarm for excessive average jitter • EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity • EVT_RTPTIMEOUT – QoS alarm for RTP inactivity • EVT_RFC2833 – RFC 2833 event
unNumOfEvents	number of events to disable (number of enumerations in **pEvents** array)
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
: Indicates that the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
: Invalid parameter

EIPM_INTERNAL
: Internal error

EIPM_INV_EVT
: Invalid event

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

EIPM_SYSTEM
: System error

EIPM_UNSUPPORTED
: Function unsupported

Example

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

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

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

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

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

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

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

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

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

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

See Also

- ipm_EnableEvents( )

ipm_EnableEvents( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pEvents	array of enumerations that specifies the events to enable The eIPM_EVENT data type is an enumeration that defines the following values: • EVT_LOSTPACKETS – QoS alarm for excessive percentage of lost packets • EVT_JITTER – QoS alarm for excessive average jitter • EVT_RTCPTIMEOUT – QoS alarm for RTCP inactivity • EVT_RTPTIMEOUT – QoS alarm for RTP inactivity • EVT_RFC2833 – RFC 2833 event

Parameter	Description
unNumOfEvents	number of events to enable (number of enumerations in **pEvents** array)
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_EVT_EXIST
Event already enabled

EIPM_EVT_LIST_FULL
Too many events

EIPM_INTERNAL
Internal error

EIPM_INV_EVT
Invalid event

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

EIPM_SYSTEM
System error

EIPM_UNSUPPORTED
Function unsupported

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

See Also

- **ipm_DisableEvents( )**

ipm_GetCTInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: ipmlib.h

Category: Media Session

Mode: Asynchronous or synchronous

Description

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

Parameter	Description
nDeviceHandle	specifies the valid IP channel handle obtained when the channel was opened using **ipm_Open( )**
pCTInfo	specifies a pointer to the CT_DEVINFO structure that contains the IP channel device information
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

Example

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

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

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

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

See Also

- **ipm_Open( )**

ipm_GetLocalMediaInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Description

The **ipm_GetLocalMediaInfo( )** function retrieves properties for the local media channel. This function retrieves the local audio and/or video 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 that the function failed.

Cautions

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

Errors

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

EIPM_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_VIDEO_LOCAL_RTP_INFO;
    // MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_UDPTL_T38_INFO;
    MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_LOCAL_RTP_INFO;
```

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

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

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

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

```
                    printf("PortId=%d\n",pMediaInfo->MediaData[i].mediaInfo.PortInfo.unPortId);
                    printf("IP=%s\n",pMediaInfo>MediaData[i].mediaInfo.PortInfo.cIPAddress);
                    break;
                }
            break;

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

See Also

None

ipm_GetParm( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

    /*
    .
    .
    . continue
    .
    .
    */
}

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

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

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

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

See Also

- **ipm_SetParm( )**

ipm_GetQoSAlarmStatus( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Description

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

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None.

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

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

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

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

See Also

- **ipm_ResetQoSAlarmStatus( )**

ipm_GetQoSThreshold( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Description

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

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

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

   /*
   .
   .
   Main Processing
   .
   .
   */

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

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

   /*
   .
   .
   . continue
   .
   .
   */
}

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

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

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

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

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

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

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

See Also

- ipm_SetQoSThreshold( )

ipm_GetSessionInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Description

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

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

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

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

See Also

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

ipm_GetXmitSlot( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

See Also

None

ipm_Listen( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates the function failed.

Cautions

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

Errors

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

EIPM_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

Description

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

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

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

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

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

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

Termination Events

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

IPMEV_ERROR
: Indicates that the function failed.

Cautions

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

Errors

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

EINVAL
: Invalid argument (system-level error)

ENOMEM
: Memory allocation failure (system-level error)

Example

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

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

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

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

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

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

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

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

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

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

See Also

- **ipm_Close( )**

ipm_ReceiveDigits( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Description

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

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pDigitInfo	pointer to data structure that contains digit information. The application must set the direction and type of digits before calling the function. On return, the function sets the unNumberOfDigits field to indicate how many IPMEV_DIGITS_RECEIVED events the application must process. See IPM_DIGIT_INFO for details.
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

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

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

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

EIPM_SYSTEM
System error

Example

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

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

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

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

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

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

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

    /*
    .
    .
     Continue processing
    .
    .
    */
}

void CheckEvent()
{

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

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

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

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

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

See Also

- **ipm_SendDigits( )**

ipm_ResetQoSAlarmStatus( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Description

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

See Also

- **ipm_GetQoSAlarmStatus( )**

ipm_SendDigits( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: I/O

Mode: asynchronous or synchronous

Description

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

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

See Also

- **ipm_ReceiveDigits( )**

ipm_SetParm( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

    /*
    .
    .
    continue
    .
    .
    */
}

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

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

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

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

See Also

- **ipm_GetParm( )**

ipm_SetQoSThreshold( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: QoS

Mode: asynchronous or synchronous

Description

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_INTERNAL
Internal error

EIPM_INV_MODE
Invalid mode

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

EIPM_SYSTEM
System error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

    /*
    .
    .
    . continue
    .
    .
    */
}

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

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

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

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

See Also

- **ipm_GetQoSThreshold( )**

ipm_SetRemoteMediaInfo( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Description

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

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

Parameter	Description
nDeviceHandle	handle of the IP Media device
pMediaInfo	media information data structure; see IPM_MEDIA_INFO for details Applications can define the following: • local transmit coder and remote transmit coder • local and remote RTP/RTCP ports • 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 that are valid for this function: • DATA_IP_RECEIVEONLY – receive data from the IP network but do not send data • DATA_IP_SENDONLY – send data to the IP network but do not receive data • DATA_IP_TDM_BIDIRECTIONAL – full duplex data path (streaming media) between IP network and TDM
usMode	operation mode Set to EV_ASYNC for asynchronous execution or to EV_SYNC for synchronous execution.

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

- The application must wait until this function completes before calling **ipm_Listen( )**.
- See IPM_AUDIO_CODER_INFO, on page 90for 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( )**

ipm_StartMedia( )

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

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Description

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

- The application must wait until this function completes before calling **ipm_Listen( )**.
- Do not set the IP address to 0.0.0.0, because this may lead to a hung port.

Errors

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

EIPM_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");

    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");

    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_STARTMEDIA for device = %s\n", ATDV_NAMEP(nDeviceID));
            break;

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

See Also

- ipm_Stop( )

ipm_Stop( )

Name: int ipm_Stop(nDeviceHandle, eOperation, usMode)

Inputs:

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

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: Media Session

Mode: asynchronous or synchronous

Description

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

None

Errors

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

EIPM_BADPARM
Invalid parameter

EIPM_FWERROR
Firmware error

Example

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

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

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

    /*
    .
    .
    Main Processing
    .
    .
    */

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

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

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

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

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

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

See Also

- **ipm_UnListen( )**

ipm_UnListen( )

Name: int ipm_UnListen(nDeviceHandle, usMode)

Inputs: int nDeviceHandle — • IP Media device handle

unsigned short usMode — • async or sync mode setting

Returns: 0 on success
-1 on failure

Includes: srllib.h
ipmlib.h

Category: System Control

Mode: asynchronous or synchronous

Description

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

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

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

Termination Events

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

IPMEV_ERROR
Indicates that the function failed.

Cautions

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

Errors

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

EIPM_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. One event is returned for each digit that is received. Event contains digit data in IPM_DIGIT_INFO data structure.

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

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

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

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

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

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

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

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

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

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

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

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

IPMEV_QOS_ALARM
Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when desired QoS alarm triggers. No data is returned in the event.

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

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

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

IPMEV_RFC2833SIGNALRECEIVED
Unsolicited event enabled via **ipm_EnableEvents( )**. Event is generated when RFC2833 signal is detected on IP. Event contains signal data in an IPM_RFC2833_SIGNALID_INFO data structure.

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

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

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

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

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

IPMEV_T38CALLSTATE
: Unsolicited event enabled via **ipm_EnableEvents( )**. Event is returned when T.38 call state changes. Event data is an eIPM_T38CALLSTATE enumeration identifying the new call state.

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

Data Structures 4

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

CT_DEVINFO

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

Description

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

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

Field Descriptions

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

ct_prodid
: contains a valid product identification number for the device

ct_devfamily
: specifies the device family; possible values are:
  - CT_DFHMPDM3 – Intel NetStructure HMP (Host Media Processing) device

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

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

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

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

ct.ext_devinfo.ct_RFU
not used

ct_ext_devinfo.ct_net_devinfo.ct_prottype
not used

Example

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

IPM_AUDIO_CODER_INFO

IPM_CODER_INFO

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

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

Description

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

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

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

Field Descriptions

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

eCoderType
: type of coder to be used for streaming media operations

  The following values are defined:

  - CODER_TYPE_G711ALAW64K – G.711, A-law, 64 kbps
  - CODER_TYPE_G711ULAW64K – G.711, mu-law, 64 kbps
  - CODER_TYPE_G7231_5_3K – G.723.1, 5.3 kbps
  - CODER_TYPE_G7231_6_3K – G.723.1, 6.3 kbps
  - CODER_TYPE_G729ANNEXA – G.729 Annex A
  - CODER_TYPE_G729ANNEXAWANNEXB – G.729 Annex A with Annex B

eFrameSize
: size of frame for coders that support multiple frame sizes—currently G.711 coders only. (All other coders have a predefined, standard value for the frame size and have a user-programmable frames per packet field in the IPM_CODER_INFO data structure.) When packets are sent in both directions (that is, when the call to **ipm_StartMedia( )** or **ipm_SetRemoteMediaInfo( )** specifies **eDirection** = DATA_IP_TDM_BIDIRECTIONAL),

the application must know the frame size of incoming packets and use eIPM_CODER_FRAMESIZE to specify that value.

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

- 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 1. This field cannot be modified for G.711 coders.

eVadEnable

flag for enabling/disabling VAD (Voice Activity Detection)

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

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

unCoderPayloadType

RTP header payload type using RFC 1890 standard definitions. The application is responsible for negotiating this value between the two endpoints. This may be set to any value for non-standard coders or if the application does not require interoperability with third-party applications. Values: 0 to 127. 96 to 127 is the dynamic range.

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

unRedPayloadType

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

Table 1. Supported Audio Coder Properties

eCoderType	Frame Size (ms)	Frames per Packet (fpp)	eVadEnable Value
CODER_TYPE_G711ALAW64K	10, 20, or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G711ULAW64K	10, 20, or 30	(fixed at 1)	Must be CODER_VAD_DISABLE
CODER_TYPE_G723_1_5_3K	(fixed at 30)	2 or 3	Either value
CODER_TYPE_G723_1_6_3K	(fixed at 30)	2 or 3	Either value
CODER_TYPE_G729ANNEXA	(fixed at 10)	2, 3, or 4	Must be CODER_VAD_DISABLE
CODER_TYPE_G729ANNEXAWANNEXB	(fixed at 10)	2, 3, or 4	Must be CODER_VAD_ENABLE [1]

NOTES:
1. Applications must explicitly enable VAD even though G.723a+b implicitly supports VAD.

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_DIGIT_INFO

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

} IPM_DIGIT_INFO, *PIPM_DIGIT_INFO;
```

Description

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

Field Descriptions

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

eDigitType
: must be set to DIGIT_ALPHA_NUMERIC

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

  - DIGIT_ALPHA_NUMERIC – alphanumeric digits

eDigitDirection
: must be set to set to DIGIT_TDM

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

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

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

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

unTimeStamp
: reserved for future use; set to 0

unExpirationTime
: reserved for future use; set to 0

unDuration
: reserved for future use; set to 0

IPM_EVENT_INFO

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

Description

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

Field Descriptions

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

unCount
: number of data structures pointed to

*pEventData
: pointer to structure containing event-specific data

IPM_FAX_SIGNAL

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

} IPM_FAX_SIGNAL, *PIPM_FAX_SIGNAL;
```

Description

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

Field Descriptions

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

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

unToneDuration
: duration of tone to generate

IPM_MEDIA

```
struct IPM_MEDIA_tag
{
   eIPM_MEDIA_TYPE eMediaType;
   union
   {
      IPM_PORT_INFO           PortInfo;          /* RTP port information     */
      IPM_AUDIO_CODER_INFO    AudioCoderInfo;    /* Audio coder information  */
      IPM_FAX_SIGNAL          FaxSignal;         /* Fax signal information   */
      IPM_VIDEO_CODER_INFO    VideoCoderInfo     /* Video coder 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_AUDIO_CODER_INFO, IPM_FAX_SIGNAL, and IPM_VIDEO_CODER_INFO. This structure is a child of the IPM_MEDIA_INFO structure which is used by the **ipm_GetLocalMediaInfo( )** and **ipm_StartMedia( )** functions.

Field Descriptions

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

eMediaType
type of media used to start an IP session

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

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

PortInfo
reference to RTP port information data structure, type IPM_PORT_INFO

AudioCoderInfo
reference to audio coder information data structure, type IPM_AUDIO_CODER_INFO

FaxSignal
reference to fax signal data structure, type IPM_FAX_SIGNAL

VideoCoderInfo
reference to video coder information data structure, type IPM_VIDEO_CODER_INFO

IPM_MEDIA_INFO

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

} IPM_MEDIA_INFO, *PIPM_MEDIA_INFO;
```

Description

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

Field Descriptions

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

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

MediaData
: reference to IPM_MEDIA structures

IPM_OPEN_INFO

Description

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

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

IPM_PARM_INFO

```
typedef struct ipm_param_info_tag
{
    eIPM_PARM     eParm;          /* the parameter to set or get */
    void          *pvParmValue;   /* pointer to value of parameter */
} IPM_PARM_INFO, *PIPM_PARM_INFO;
```

Description

This structure is used to set or retrieve parameters for an IP channel. The structure is used by the **ipm_GetParm( )** and **ipm_SetParm( )** functions.

Field Descriptions

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

eIPM_PARM
type of parameter to set or get. See Table 2 for supported types and corresponding values.

pvParmValue
pointer to the value of the parameter. See Table 2 for supported values for each parameter type.

Table 2. eIPM_PARM Parameters and Values

eIP_PARM Define	Description and Values
PARMCH_DTMFXFERMODE	Sets DTMF transfer mode Type: eIPM_DTMFXFERMODE (enumeration). Values: • DTMFXFERMODE_INBAND – in-band (default) • DTMFXFERMODE_OUTOFBAND – out-of-band • DTMFXFERMODE_RFC2833 – RFC2833 **Note:** In order for DTMF event reporting to occur, you must set out-of-band signaling on the receive side.
PARMCH_RFC2833EVT_RX_PLT	RFC2833 event receive payload type Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMCH_RFC2833EVT_TX_PLT	RFC2833 event transmit payload type Type: unsigned char. Valid values: 96 to 127. Default: 101.
PARMCH_RX_ADJVOLUME	Indicates volume level adjustment for inbound (from IP) side in 1 dB increments. Type: integer. Valid values: -32 to 31. Default: 0 (no adjustment).
PARMCH_TX_ADJVOLUME	Indicates volume level adjustment for outbound (to IP) side in 1 dB increments. Type: integer. Valid values: -32 to 31. Default: 0 (no adjustment).
PARMCH_TOS	Indicates type of service in IPv4 headers. This can be either a 7-bit TOS field or a 6-bit DSCP field for Differentiated Services per RFC2474[. Type: char. Valid values: 0 to 255. Default: 0.
PARMCH_TTL	Set time-to-live for multicast. Type: char. Valid values: 0 to 255. Default: 1.

IPM_PORT_INFO

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

} IPM_PORT_INFO, *PIPM_PORT_INFO;
```

Description

This structure contains RTP, RTCP, and T.38 UDP port properties. It is a child of IPM_MEDIA, which is a child of the IPM_MEDIA_INFO structure that is used by **ipm_GetLocalMediaInfo( )** and **ipm_StartMedia( )**.

Field Descriptions

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

unPortId
: port identifier

cIPAddress[IP_ADDR_SIZE]
: null-terminated IP address of the port in standard dotted decimal string format; for example, 192.168.0.1

  Note: Avoid setting IP address 0.0.0.0 when using **ipm_StartMedia( )** because this may cause a hung port.

IPM_QOS_ALARM_DATA

```
typedef struct ipm_qos_alarm_data_tag
{
   eIPM_QOS_TYPE     eQoSType;          /* The QOS parameter type  */
   eIPM_ALARM_STATE  eAlarmState;       /* indicate if On/Off */

} IPM_QOS_ALARM_DATA, *PIPM_QOS_ALARM_DATA;
```

Description

This structure is used to retrieve data associated with QoS alarms, as reported in IPMEV_QOS_ALARM events. It is also a child of the IPM_QOS_ALARM_STATUS structure, which is used by the **ipm_GetQoSAlarmStatus( )** and **ipm_ResetQoSAlarmStatus( )** functions.

The library generates a IPMEV_QOS_ALARM alarm event with ALARM_STATE_ON when a QoS fault threshold is exceeded, and it generates a generates a QoS alarm event with ALARM_STATE_OFF when the fault measurement returns to a sub-threshold level.

Field Descriptions

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

eQoSType
identifies the alarm event that has occurred

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

- QOSTYPE_JITTER – QoS alarm for excessive average jitter
- QOSTYPE_LOSTPACKETS – excessive lost packets
- QOSTYPE_RTCPTIMEOUT – RTCP inactivity
- QOSTYPE_RTPTIMEOUT – RTP inactivity

eAlarmState
alarm on / off flag

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

- ALARM_STATE_OFF – alarm is OFF
- ALARM_STATE_ON – alarm is ON

IPM_QOS_ALARM_STATUS

```
typedef struct ipm_qos_alarm_status_tag
{
   unsigned int unAlarmCount;
   IPM_QOS_ALARM_DATA QoSData[MAX_ALARM];

} IPM_QOS_ALARM_STATUS, *PIPM_QOS_ALARM_STATUS;
```

Description

This structure contains the status of QoS alarms for an IP channel. It is the parent of IPM_QOS_ALARM_DATA and is used by **ipm_GetQoSAlarmStatus( )** and **ipm_ResetQoSAlarmStatus( )**.

Field Descriptions

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

unAlarmCount
: number of QoSData structures to follow
: maximum number of alarms = MAX_ALARM

QoSData
: reference to alarm data information structure IPM_QOS_ALARM_DATA

IPM_QOS_SESSION_INFO

```
typedef struct ipm_qos_session_info_tag
{
      eIPM_QOS_TYPE  eQoSType;
      unsigned int   unData;

} IPM_QOS_SESSION_INFO, *PIPM_QOS_SESSION_INFO;
```

Description

This structure reports statistical Quality of Service information for an IP session. It is a child of the IPM_SESSION_INFO structure which is filled in when **ipm_GetSessionInfo( )** returns successfully.

Field Descriptions

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

eQoSType
identifies the QoS alarm to retrieve statistics for

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

- QOSTYPE_JITTER – average jitter (in msec) since beginning of call
- QOSTYPE_LOSTPACKETS – percentage of lost packets since beginning of call
- QOSTYPE_RTCPTIMEOUT – RTCP inactivity (in msec)
- QOSTYPE_RTPTIMEOUT – RTP inactivity (in msec)

unData
value of the QoS parameter

IPM_QOS_THRESHOLD_DATA

```
typedef struct ipm_qos_threshold_data_tag
{
    eIPM_QOS_TYPE eQoSType;
    unsigned int  unTimeInterval;
    unsigned int  unDebounceOn;
    unsigned int  unDebounceOff;
    unsigned int  unFaultThreshold;
    unsigned int  unPercentSuccessThreshold;
    unsigned int  unPercentFailThreshold;

} IPM_QOS_THRESHOLD_DATA, *PIPM_QOS_THRESHOLD_DATA;
```

Description

This structure contains the threshold values for QoS alarms for an IP channel. It is a child of the IPM_QOS_THRESHOLD_INFO structure which is used by **ipm_GetQoSThreshold( )** and **ipm_SetQoSThreshold( )**. When enabling a QoS alarm, default threshold and timing values as shown in Table 3 will be used unless **ipm_SetQoSThreshold( )** is used to set non-default values. Note that when an application sets a specific value for any field of a IPM_QOS_THRESOLD structure, it must explicitly set *all* fields in the structure even when default values are desired for some of the fields.

Field Descriptions

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

eQoSType
: type of QoS parameter to measure

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

  - QOSTYPE_JITTER – jitter
  - QOSTYPE_LOSTPACKETS – lost packets
  - QOSTYPE_RTCPTIMEOUT – RTCP inactivity
  - QOSTYPE_RTPTIMEOUT – RTP inactivity

unTimeInterval
: time interval (in ms) between successive parameter measurements. Value should be set to a multiple of 100; other values are rounded to the nearest hundred.

  Note: Value must be greater than unFaultThreshold for the jitter QoS type.

unDebounceOn
: time interval for detecting potential alarm fault condition. Must be set to a value that is a multiple of unTimeInterval; other values are rounded down to the next lower multiple of unTimeInterval.

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

unDebounceOff
: time interval for detecting potential alarm non-fault condition. Must be set to a value that is a multiple of unTimeInterval; other values are rounded down to the next lower multiple of unTimeInterval.

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

unFaultThreshold

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

- QOSTYPE_JITTER – allowable average jitter, in ms. Range: 0 to 1000 (ms)
- QOSTYPE_LOSTPACKET – allowable percentage of lost packets. Range: 0 to 100 (%)
- QOSTYPE_RTCPTIMEOUT – allowable RTCP inactive interval before an alarm is sent, in units of 100 ms. Range: 50 to 1200 (x100 ms)
- QOSTYPE_RTPTIMEOUT – allowable RTP inactive interval before an alarm is sent, in units of 100 ms. Range: 50 to 1200 (x100 ms)

unPercentSuccessThreshold

percentage of poll instances in unDebounceOff time interval that the fault threshold must not be exceeded before an "alarm off" event is sent. Allowed values correspond to multiples of the ratio of unDebounceOff to unTimeInterval (i.e., the inverse of the number of poll instances) expressed as an integer percentage; other values are truncated to the next lower percentage multiple.

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

unPercentFailThreshold

percentage of poll instances in unDebounceOn time interval that the fault threshold must be exceeded before an "alarm on" event is sent. Allowed values correspond to multiples of the ratio of unDebounceOn to unTimeInterval (i.e., the inverse of the number of poll instances) expressed as a integer percentage; other values are truncated to the next lower percentage multiple.

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

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

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold [1]	% Success Threshold	% Fail Threshold
Jitter	5000	20000	60000	60 (ms)	25	25
Lost Packets	1000	10000	10000	20 (%)	40	40
RTCP Timeout	1000	0	0	250 (x100ms = 25sec)	0	0
RTP Timeout	1000	0	0	1200 (x100ms = 120sec)	0	0

Notes:
1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.

QoS debouncing is calculated as an integer number of parameter measurements that must exceed (or fall below) the fault threshold within the debounce interval before an alarm-on (or alarm-off) event is generated. The calculation uses the following formulas:

For QoS alarm-on debouncing:

count = int(int(unDebounceOn/unTimeInterval) * (unPercentFailThreshold/100))

For QoS alarm-off debouncing:

count = int(int(unDebounceOff/unTimeInterval) * (unPercentSuccessThreshold/100))

IPM_QOS_THRESHOLD_INFO

```
typedef struct ipm_qos_threshold_info_tag
{
   unsigned int unCount;
   IPM_QOS_THRESHOLD_DATA QoSThresholdData[MAX_QOS_THRESHOLD];

} IPM_QOS_THRESHOLD_INFO, *PIPM_QOS_THRESHOLD_INFO;
```

Description

This structure is used to set and get the threshold values for QoS alarms for a single IP channel. It is the parent of IPM_QOS_THRESHOLD_DATA and is used by **ipm_GetQoSThreshold( )** and **ipm_SetQoSThreshold( )**.

Field Descriptions

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

unCount
: number of IPM_QOS_THRESHOLD_DATA structures to follow;
maximum = MAX_QOS_THRESHOLD

QosThresholdData
: array of structures containing alarm trigger settings

IPM_RFC2833_SIGNALID_INFO

```
typedef struct ipm_rfc2833_signalid_info_tag
{
   eIPM_RFC2833_SIGNAL_ID  eSignalID;
   eIPM_SIGNAL_STATE       eState;

} IPM_RFC2833_SIGNALID_INFO;
```

Description

This structure is used to identify RFC 2833-compliant signals that are received from the IP network. It is used in the IPMEV_RFC2833SIGNALRECEIVED events that are generated when RFC2833 signals are detected on IP.

Field Descriptions

The fields of the IPM_RFC2833_SIGNALID_INFO data structure are described as follows:

eSignalID
: Identifies the RFC2833 signal. The eIPM_RFC2833_SIGNAL_ID is an enumeration with values as listed in Table 4.

Table 4. eIPM_RFC2833_SIGNAL_ID Values

Name	Value (Hex)
SIGNAL_ID_EVENT_DTMF_0	0x0
SIGNAL_ID_EVENT_DTMF_1	0x1
SIGNAL_ID_EVENT_DTMF_2	0x2
SIGNAL_ID_EVENT_DTMF_3	0x3
SIGNAL_ID_EVENT_DTMF_4	0x4
SIGNAL_ID_EVENT_DTMF_5	0x5
SIGNAL_ID_EVENT_DTMF_6	0x6
SIGNAL_ID_EVENT_DTMF_7	0x7
SIGNAL_ID_EVENT_DTMF_8	0x8
SIGNAL_ID_EVENT_DTMF_9	0x9
SIGNAL_ID_EVENT_DTMF_STAR	0xa
SIGNAL_ID_EVENT_DTMF_POUND	0xb
SIGNAL_ID_EVENT_DTMF_A	0xc
SIGNAL_ID_EVENT_DTMF_B	0xd
SIGNAL_ID_EVENT_DTMF_C	0xe
SIGNAL_ID_EVENT_DTMF_D	0xf

eState
: This field is not used when receiving RFC2833 signals.

IPM_RTCP_SESSION_INFO

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

} IPM_RTCP_SESSION_INFO, *PIPM_RTCP_SESSION_INFO;
```

Description

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

Field Descriptions

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

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

unLocalSR_TxPackets
: number of packets sent by the local sender

unLocalSR_TxOctets
: number of bytes sent by the local sender

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

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

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

unLocalRR_SeqNumber
: last sequence number received from the local receiver

unLocalRR_ValidInfo
: reserved for future use

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

unRemoteSR_TxPackets
number of packets sent by the remote sender

unRemoteSR_TxOctets
number of bytes sent by the remote sender

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

- FALSE
- TRUE

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

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

unRemoteRR_SeqNumber
last sequence number received from the remote receiver

unRemoteRR_ValidInfo
reserved for future use

IPM_SESSION_INFO

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

} IPM_SESSION_INFO, *PIPM_SESSION_INFO;
```

Description

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

Field Descriptions

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

RtcpInfo
: reference to RTCP session information structure IPM_RTCP_SESSION_INFO

unQoSInfoCount
: number of IPM_QOS_SESSION_INFO structures to follow;
maximum sessions = MAX_QOS_SESSION

QoSInfo
: reference to QoS session information structure IPM_QOS_SESSION_INFO

IPM_VIDEO_CODER_INFO

```
typedef struct ipm_video_coder_info_tag
{
   unsigned int           unVersion;           /* structure version                    */
   eIPM_CODER_TYPE        eCoderType;          /* The coder Type                       */
   unsigned int           unFrameRate;         /* Frame rate, fps x100                 */
   unsigned int           unSamplingRate;      /* Sampling rate                        */
   unsigned int           unCoderPayloadType;  /* Type of coder payload supported      */
   unsigned int           unProfileLevelID;    /* Profile and level IDs                */
   char*                  szVisualConfiguration;/* Representation of visual configuration */

} IPM_VIDEO_CODER_INFO, *PIPM_VIDEO_CODER_INFO;
```

Description

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

Field Descriptions

The fields of the IPM_VIDEO_CODER_INFO data structure are described as follows.

unVersion
: version number of the data structure. Applications should *not* manipulate this field.

eCoderType
: type of coder to be used for streaming media operations. The following values are defined:
  - CODER_TYPE_H263 – H.263 video coder

unFrameRate
: video frame rate specified as 100x the actual number of frames per second. The following values are valid:
  - 600 – frame rate is 6 fps
  - 1000 – frame rate is 10 fps
  - 1500 – frame rate is 15 fps

unSamplingRate
: sampling rate of media stream. This value is typically set to 90,000.

unCoderPayloadType
: RTP header payload type using RFC 1890 standard definitions. The application is responsible for negotiating this value between the two endpoints. This may be set to any value for non-standard coders or if the application does not require interoperability with third-party applications. Values: 0 to 127; 96 to 127 is the dynamic payload range.

unProfileLevelID
: profile and level IDs

szVisualConfiguration
: hexadecimal octets for representing the visual configuration. Refer to RFC 2190 (RTP Payload Format for H.263 Video Streams) for more information.

SC_TSINFO

```
typedef struct sc_tsinfo {
   unsigned long   sc_numts;
   long            *sc_tsarrayp;
} SC_TSINFO;
```

Description

This structure defines the TDM bus (CT Bus) time slot information. It is used by **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_StartMedia( )**, and **ipm_GetLocalMediaInfo( )**.

Field Descriptions

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

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

sc_tsarrayp
: time slot ID number

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_SetParm( )**, **ipm_Stop( )**, **ipm_UnListen( )**

EIPM_INTERNAL
: Internal error. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_SetRemoteMediaInfo( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_INTERNAL_INIT
: Internal initialization error.

EIPM_INV_DEVNAME
: Invalid device name.

EIPM_INV_EVT
Invalid event. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**

EIPM_INV_MODE
Invalid mode. **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_SetRemoteMediaInfo( )**, **ipm_StartMedia( )**

EIPM_INV_STATE
Invalid state. Error indicates that initial command did not complete before another function call was made. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_SetRemoteMediaInfo( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_NOERROR
No error.

EIPM_NOMEMORY
Memory allocation error.

EIPM_RESOURCEINUSE
Resource in use or not available.

EIPM_SRL
SRL error.

EIPM_SRL_SYNC_TIMEOUT
SRL timeout.

EIPM_SYSTEM
System error. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**, **ipm_GetLocalMediaInfo( )**, **ipm_GetQoSAlarmStatus( )**, **ipm_GetQoSThreshold( )**, **ipm_GetSessionInfo( )**, **ipm_GetXmitSlot( )**, **ipm_Listen( )**, **ipm_ReceiveDigits( )**, **ipm_ResetQoSAlarmStatus( )**, **ipm_SendDigits( )**, **ipm_SetQoSThreshold( )**, **ipm_SetRemoteMediaInfo( )**, **ipm_StartMedia( )**, **ipm_UnListen( )**

EIPM_TIMEOUT
Timeout.

EIPM_UNSUPPORTED
Function unsupported. **ipm_DisableEvents( )**, **ipm_EnableEvents( )**

Glossary

Codec: see COder/DECoder

COder/DECoder: A circuit used to convert analog voice data to digital and digital voice data to analog audio.

Computer Telephony (CT): Adding computer intelligence to the making, receiving, and managing of telephone calls.

DTMF: Dual-Tone Multi-Frequency

Dual-Tone Multi-Frequency: A way of signaling consisting of a push-button or touch-tone dial that sends out a sound consisting of two discrete tones that are picked up and interpreted by telephone switches (either PBXs or central offices).

Emitting Gateway: called by a G3FE. It initiates IFT service for the calling G3FE and connects to a Receiving Gateway.

E1: The 2.048 Mbps digital carrier system common in Europe.

FCD file: An ASCII file that lists any non-default parameter settings that are necessary to configure a DM3 hardware/firmware product for a particular feature set. The downloader utility reads this file, and for each parameter listed generates and sends the DM3 message necessary to set that parameter value.

Frame: A set of SCbus/CT Bus timeslots which are grouped together for synchronization purposes. The period of a frame is fixed (at 125 µsec) so that the number of time slots per frame depends on the SCbus/CT Bus data rate.

G3FE: Group 3 Fax Equipment. A traditional fax machine with analog PSTN interface.

Gatekeeper: An H.323 entity on the Internet that provides address translation and control access to the network for H.323 Terminals and Gateways. The Gatekeeper may also provide other services to the H.323 terminals and Gateways, such as bandwidth management and locating Gateways.

Gateway: A device that converts data into the IP protocol. It often refers to a voice-to-IP device that converts an analog voice stream, or a digitized version of the voice, into IP packets.

H.323: A set of International Telecommunication Union (ITU) standards that define a framework for the transmission of real-time voice communications through Internet protocol (IP)-based packet-switched networks. The H.323 standards define a gateway and a gatekeeper for customers who need their existing IP networks to support voice communications.

IAF: Internet Aware Fax. The combination of a G3FE and a T.38 gateway.

IFP: Internet Facsimile Protocol

IFT: Internet Facsimile Transfer

International Telecommunications Union (ITU): An organization established by the United Nations to set telecommunications standards, allocate frequencies to various uses, and hold trade shows every four years.

Internet: An inter-network of networks interconnected by bridges or routers. LANs described in H.323 may be considered part of such inter-networks.

Internet Protocol (IP): The network layer protocol of the transmission control protocol/Internet protocol (TCP/IP) suite. Defined in STD 5, Request for Comments (RFC) 791. It is a connectionless, best-effort packet switching protocol.

Internet Service Provider (ISP): A vendor who provides direct access to the Internet.

Internet Telephony: The transmission of voice over an Internet Protocol (IP) network. Also called Voice over IP (VoIP), IP telephony enables users to make telephone calls over the Internet, intranets, or private Local Area Networks (LANs) and Wide Area Networks (WANs) that use the Transmission Control Protocol/Internet Protocol (TCP/IP).

ITU: See International Telecommunications Union.

Jitter: The deviation of a transmission signal in time or phase. It can introduce errors and loss of synchronization in high-speed synchronous communications.

NIC (Network Interface Card): Adapter card inserted into computer that contains necessary software and electronics to enable a station to communicate over network.

PCD file: An ASCII text file that contains product or platform configuration description information that is used by the DM3 downloader utility program. Each of these files identifies the hardware configuration and firmware modules that make up a specific hardware/firmware product. Each type of DM3-based product used in a system requires a product-specific PCD file.

PSTN: see Public Switched Telephone Network

Public Switched Telephone Network: The telecommunications network commonly accessed by standard telephones, key systems, Private Branch Exchange (PBX) trunks and data equipment.

Reliable Channel: A transport connection used for reliable transmission of an information stream from its source to one or more destinations.

Reliable Transmission: Transmission of messages from a sender to a receiver using connection-mode data transmission. The transmission service guarantees sequenced, error-free, flow-controlled transmission of messages to the receiver for the duration of the transport connection.

RTCP: Real Time Control Protocol

RTP: Real Time Protocol

SIP: Session Initiation Protocol: an Internet standard specified by the Internet Engineering Task Force (IETF) in RFC 3261. SIP is used to initiate, manage, and terminate interactive sessions between one or more users on the Internet.

T1: A digital transmission link with a capacity of 1.544 Mbps used in North America. Typically channeled into 24 digital subscriber level zeros (DS0s), each capable of carrying a single voice conversation or data stream. T1 uses two pairs of twisted pair wires.

TCP: see Transmission Control Protocol

Terminal: An H.323 Terminal is an endpoint on the local area network which provides for real-time, two-way communications with another H.323 terminal, Gateway, or Multipoint Control Unit. This communication consists of control, indications, audio, moving color video pictures, and/or data between the two terminals. A terminal may provide speech only, speech and data, speech and video, or speech, data, and video.

Transmission Control Protocol: The TCP/IP standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection.

UDP: see User Datagram Protocol

UDPTL: Facsimile UDP Transport Layer protocol

User Datagram Protocol: The TCP/IP standard protocol that allows an application program on one machine to send a datagram to an application program on another machine. Conceptually, the important difference between UDP datagrams and IP datagrams is that UDP includes a protocol port number, allowing the sender to distinguish among multiple destinations on the remote machine.

VAD: Voice Activity Detection

Index