



# IP Media Library API for Linux and Windows Operating Systems

Programming Guide

---

*November 2003*



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

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

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

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

Copyright © 2002-2003 Intel Corporation. All Rights Reserved.

AnyPoint, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel Centrino logo, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, RemoteExpress, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside, TokenExpress, VoiceBrick, VTune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Publication Date: November 2003

Document Number: 05-1834-004

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

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

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

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



# Contents

---

	<b>Revision History</b> .....	6
	<b>About This Publication</b> .....	7
	Purpose .....	7
	Intended Audience .....	7
	How to Use This Publication .....	7
	Related Information .....	8
<b>1</b>	<b>Product Description</b> .....	9
	1.1 Features .....	9
	1.2 Architecture .....	9
	1.3 Introduction to the IP Media Library .....	10
	1.4 Relationship with Global Call Library .....	10
	1.5 Standard Runtime Library Support .....	11
	1.6 Media Channel Device Naming .....	11
<b>2</b>	<b>Programming Models</b> .....	13
<b>3</b>	<b>State Models</b> .....	15
<b>4</b>	<b>Event Handling</b> .....	17
	4.1 SRL Event Management Functions .....	17
	4.2 SRL Standard Attribute Functions .....	17
<b>5</b>	<b>Error Handling</b> .....	19
<b>6</b>	<b>Application Development Guidelines</b> .....	21
	6.1 Introduction to DTMF Handling .....	21
	6.2 Setting DTMF Parameters .....	22
	6.2.1 DTMF Modes .....	22
	6.2.2 Setting In-Band Mode .....	23
	6.2.3 Setting RFC 2833 Mode .....	24
	6.2.4 Setting Out-of-Band Mode .....	25
	6.3 Notification of DTMF Detection .....	26
	6.4 Generating DTMF .....	27
	6.5 Using T.38 Fax Gateway .....	27
<b>7</b>	<b>Quality of Service (QoS)</b> .....	31
	7.1 QoS Overview .....	31
	7.2 QoS Alarm Types and Thresholds .....	31
	7.3 Alarm and Recovery Mechanisms .....	32
	7.4 Using QoS Alarms .....	34
	7.5 Hints for QoS Alarm Handling .....	35
<b>8</b>	<b>Building Applications</b> .....	37
	8.1 Compiling and Linking under Linux .....	37
	8.1.1 Include Files .....	37



8.1.2	Required Libraries .....	37
8.2	Compiling and Linking under Windows .....	38
8.2.1	Include Files .....	38
8.2.2	Required Libraries .....	38
<b>Index</b>	.....	<b>39</b>

# Figures

---

1	IP Media Architecture .....	10
2	IP Media Channel State Diagram .....	15
3	In-Band Mode Scenario Diagram .....	23
4	RFC 2833 Scenario Diagram.....	25
5	Out-of-Band Mode Scenario Diagram.....	26
6	Using T.38 Fax Scenario Diagram.....	29
7	Alarm Mechanism .....	33
8	Recovery Mechanism .....	33



## Revision History

---

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

Document No.	Publication Date	Description of Revisions
05-1834-004	November 2003	<a href="#">Application Development Guidelines</a> : hid section - Using the T.38 Fax Server (for HMP 1.1 only, restored and renamed section Using T.38 Fax Gateway (for SR 6.0 PCI only)
05-1834-003	September 2003	<a href="#">Application Development Guidelines</a> : Added section - Using the T.38 Fax Server (for HMP 1.1 only), hid Using T.38 Fax (no supported in HMP 1.1) (PTR 28331) <a href="#">DTMF Modes</a> : added range of values (96-127) for PARMCH_RFC2833_EVT_TX_PLT and PARMCH_RFC2833_EVT_RX_PLT
05-1834-002	November 2002	<a href="#">Application Development Guidelines</a> : Added section - Using T.38 Fax
05-1834-001	September 2002	Initial version of this document



# About This Publication

---

The following topics provide information about this publication:

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

## Purpose

This guide provides programming guidelines for the IP media software, which is typically used with the Global Call call control application programming interface (API). This is a companion guide to the *IP Media Library API Library Reference*, which provides details on functions and parameters in the IP media software.

## Intended Audience

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

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

## How to Use This Publication

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

The information in this guide is organized as follows:

- [Chapter 1, “Product Description”](#) introduces the IP media software and its key features.
- [Chapter 2, “Programming Models”](#) describes methods of developing IP media-based applications.

- [Chapter 3, “State Models”](#) describes a simple state-based IP media application.
- [Chapter 4, “Event Handling”](#) defines an event and describes how to handle an event.
- [Chapter 5, “Error Handling”](#) presents information on how to obtain error codes and handle errors.
- [Chapter 6, “Application Development Guidelines”](#) provides information on developing IP media-based applications.
- [Chapter 7, “Quality of Service \(QoS\)”](#) details how QoS may be used in an application.
- [Chapter 8, “Building Applications”](#) describes how to compile and link IP media-based applications.

## Related Information

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

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



This chapter provides an overview of the IP media software. It contains the following sections:

- Features ..... 9
- Architecture ..... 9
- Introduction to the IP Media Library ..... 10
- Relationship with Global Call Library ..... 10
- Standard Runtime Library Support ..... 11
- Media Channel Device Naming ..... 11

## 1.1 Features

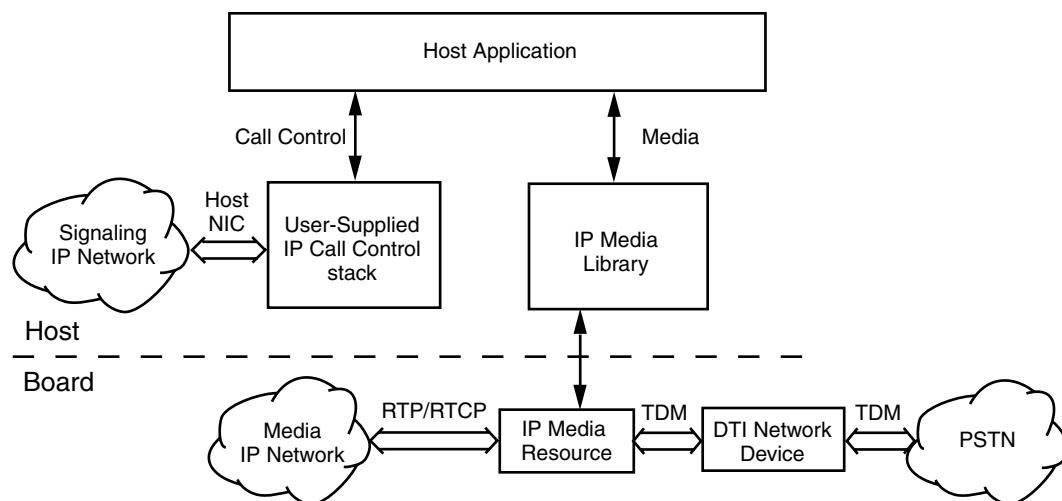
Some of the features of the IP media software include:

- media resource management, such as open, close, and configure tasks
- media resource operations, such as start, stop, and detect digits
- Quality of Service (QoS) threshold alarm configuration and status reporting
- support of standard runtime library event management routines for error retrieval
- compatibility with Global Call or another call control stack to provide IP call control functionality

## 1.2 Architecture

Figure 1 shows the IP media library architecture when using an Intel® NetStructure DM/IP board or an Intel® NetStructure IPT board and a user-supplied call control stack.

Figure 1. IP Media Architecture



### 1.3 Introduction to the IP Media Library

The IP media library (IPML) provides an application programming interface to control the starting and stopping of RTP sessions, transmit and receive DTMF or signals, QoS alarms and their thresholds, and general-purpose device control functions. The library is only used to control media functions. It is not used to control the signaling stack. The application developer may choose to integrate any third party IP signaling stack (H.323, SIP, MGCP, etc.), or implement a proprietary signaling stack solution. The application developer uses the IP signaling stack to initiate or answer calls, and negotiate media characteristics such as coder, frames per packet, destination IP address, etc. Once media characteristics have been negotiated, the application uses IPML functions to start RTP streaming using the desired media characteristics.

### 1.4 Relationship with Global Call Library

The Global Call library provides a common call control interface that is independent of the underlying network interface technology. While the Global Call library is primarily used for call establishment and teardown, it also provides capabilities to support applications that use IP technology, such as:

- call control capabilities for establishing calls over an IP network, via the RADVISION H.323 and SIP signaling stacks
- support for IP media control by providing the ability to open and close IP media channels for streaming, using the IP media software internally (under the hood)

**Note:** Applications should not mix Global Call and IP media library usage of the same ipm\_ devices.

Refer to the following Global Call manuals for more details:

- *Global Call IP Technology User's Guide*

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

## 1.5 Standard Runtime Library Support

The IP media library performs event management using the Standard Run-time Library (SRL), which provides a set of common system functions that are applicable to all devices. SRL functions, parameters, and data structures are described in the *Standard Runtime Library API Library Reference*. Use the SRL functions to simplify application development by writing common event handlers to be used by all devices.

## 1.6 Media Channel Device Naming

To determine available resources, call **ipm\_Open()** on a board device, then call **ATDV\_SUBDEVS** to get the available resources. (SRL operations are described in the *Standard Runtime Library API Library Reference*.)

To determine available resources in the Windows environment, use the **sr\_getboardcnt()** function, which returns the number of boards of a particular type. (SRL operations are described in the *Standard Runtime Library API Library Reference*.)

Each IP media channel device follows the naming convention **ipmBxCy**; where:

- B is followed by the unique logical board number
- C is followed by the number of the media device channel

You may also use the **ipm\_Open()** function to open a board device, **ipmBx**, where B is followed by the unique logical board number.

Before you can use any of the other IP media library functions on a device, that device must be opened. When the device is opened using **ipm\_Open()**, the function returns a unique device handle. The handle is the only way the device can be identified once it has been opened. The **ipm\_Close()** function closes a device.



This chapter describes the programming models supported by the IP media software.

The *Standard Runtime Library API Programming Guide* describes different programming models which can be used by applications. The IP media library supports all the programming models described therein.

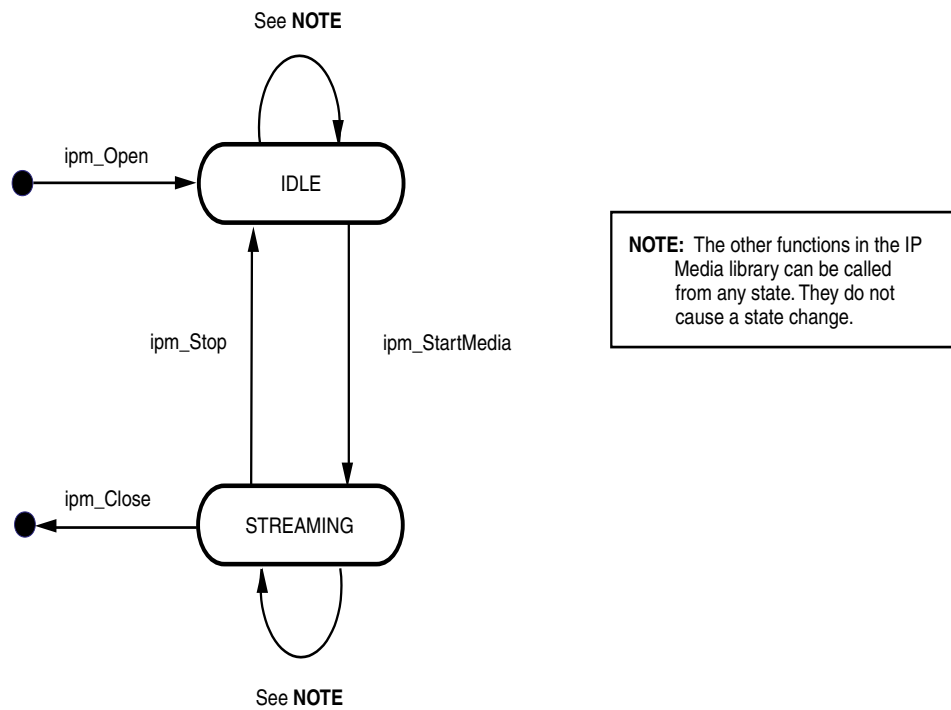
**Note:** The synchronous programming model is recommended for low density systems only. For high density systems, asynchronous programming models provide increased throughput for the application.



This chapter describes a very simple IP media state-based application.

Figure 2 shows a simple IP media application using two channel device states, IDLE and STREAMING.

**Figure 2. IP Media Channel State Diagram**







All IP media events are retrieved using standard runtime library (SRL) event retrieval mechanisms, including event handlers. The SRL is a device-independent library containing Event Management functions and Standard Attribute functions. This chapter lists SRL functions that are typically used by IP media-based applications.

- [SRL Event Management Functions](#) . . . . . 17
- [SRL Standard Attribute Functions](#) . . . . . 17

## 4.1 SRL Event Management Functions

SRL Event Management functions retrieve and handle device termination events for certain library functions. Applications typically use the following functions:

- sr\_enbhdlr()**  
enables event handler
- sr\_dishdlr()**  
disables event handler
- sr\_getevtdev()**  
gets device handle
- sr\_getevttype()**  
gets event type
- sr\_waitevt()**  
wait for next event
- sr\_waitevtEx()**  
wait for events on certain devices

*Note:* See the *Standard Runtime Library API Library Reference* for function details.

## 4.2 SRL Standard Attribute Functions

SRL Standard Attribute functions return general device information, such as the device name or the last error that occurred on the device. Applications typically use the following functions:

- ATDV\_ERRMSGP()**  
pointer to string describing the error that occurred during the last function call on the specified device
- ATDV\_LASTERR()**  
error that occurred during the last function call on a specified device. See the function description for possible errors for the function.

**ATDV\_NAMEP()**

pointer to device name, for example, ipmBxCy

**ATDV\_SUBDEVS()**

number of subdevices

**Note:** See the *Standard Runtime Library API Library Reference* for function details.

This chapter describes error handling for the IP media software.

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 standard runtime library functions **ATDV\_LASTERR()** and **ATDV\_ERRMSGP()**.



This chapter contains guidelines for developing applications which use the IP media library. The following topics are discussed:

- [Introduction to DTMF Handling](#) ..... 21
- [Setting DTMF Parameters](#) ..... 22
- [Notification of DTMF Detection](#) ..... 26
- [Generating DTMF](#) ..... 27
- [Using T.38 Fax Gateway](#) ..... 27

## 6.1 Introduction to DTMF Handling

When a session is started on an IPM device, the IPM device receives data from its IP interface and transmits data towards the TDM bus. A DTI device receives data from its PSTN interface and transmits towards the TDM bus as well. In a gateway configuration, the DTI and IPM devices will be configured, via `gc_Listen()` and `ipm_Listen()` respectively, to listen to each other and thus create a full duplex communication path. The IPM device will forward DTMF that it receives on one interface to the other interface. [Figure 1, “IP Media Architecture”](#), on page 10 shows the data flow between the IP media library, the IP network, and the PSTN network.

When an IPM device receives DTMF from the TDM bus, there are several ways to forward it towards the IP interface. These include: forwarding it in the RTP stream (also called in-band), sending via RFC 2833, and using an application-controlled/defined method (also called out-of-band).

The IPM device can automatically forward the DTMF when either the in-band or RFC 2833 DTMF transfer mode has been selected. DTMF is **not** automatically forwarded when the application controlled/defined method, also known as out-of-band mode, has been selected. In the out-of-band case, the application must call `ipm_ReceiveDigits()` and have an `IPM_DIGITS_RECEIVED` event handler in place. Upon receiving the `IPM_DIGITS_RECEIVED` event, the DTMF information is contained in the `IPM_DIGIT_INFO` structure delivered with the event. The application has the responsibility to forward the DTMF via whatever mechanism, open or proprietary, it desires.

When using RFC 2833 mode, the DTMF could optionally be sent in both RFC 2833 packets and in-band. The default is that the DTMF is only sent in the RFC 2833 packet and the audio is muted. Setting the mute audio parameter (`IPM_RFC2833MUTE_AUDIO`) to `RFC2833MUTE_AUDIO_OFF` will cause the DTMF to be sent in-band as well as via RFC 2833.

**Note:** Use caution when using both RFC 2833 and in-band DTMF, because an endpoint device may recognize two separate digits instead of one.

When using out-of-band mode, the DTMF is never transmitted in-band. As mentioned earlier, the application has the responsibility to forward the digits.

The setting for DTMF transfer mode also affects the handling of DTMF that is received from the IP interface. When the mode is set to in-band, the DTMF is automatically forwarded to the TDM bus.

If out-of-band mode has been selected, then the application will use its own mechanism to be notified that a DTMF digit has been received. Then, **ipm\_SendDigits()** is used when necessary to transmit a DTMF digit towards the TDM bus.

When the mode is set to RFC 2833, DTMF is automatically forwarded to the TDM bus as PCM data.

**Note:** For Intel® NetStructure™ DM/IP Series boards only: if you wish to be notified of RFC 2833 packets as they arrive at the IP port, the application must enable the EVT\_RFC2833 event via a call to **ipm\_EnableEvents()**. Upon receiving the IPMEV\_RFC2833SIGNALRECEIVED event, the DTMF information is contained in the IPM\_RFC2833\_SIGNALID\_INFO structure. The application must use **ipm\_SendDigits()** to forward the digit towards the TDM bus.

## 6.2 Setting DTMF Parameters

This section contains the following topics:

- [DTMF Modes](#)
- [Setting In-Band Mode](#)
- [Setting RFC 2833 Mode](#)
- [Setting Out-of-Band Mode](#)

### 6.2.1 DTMF Modes

The IP media library can be used to configure which DTMF mode (in-band, RFC 2833, or out-of-band) is used by the application. The DTMF mode is set on a per-channel basis using **ipm\_SetParm()** and the IPM\_PARM\_INFO data structure.

The eIPM\_DTMFXFERMODE enumeration identifies which DTMF mode to use. The following values are supported:

- DTMFXFERMODE\_INBAND - DTMF digits are sent and received in-band via standard RTP transcoding. This is the default mode when a channel is opened.
- DTMFXFERMODE\_RFC2833 - DTMF digits are sent and received in the RTP stream as defined in RFC 2833.
- DTMFXFERMODE\_OUTOFBAND - DTMF digits are sent and received outside the RTP stream.

Depending on the mode being used, the digit information transferred in the RTP stream.

When using RFC2833, the payload type is specified by using the following parameter/value setting in a call to **ipm\_SetParm()**:

- PARMCH\_RFC2833\_EVT\_TX\_PLT - Identifies the transmit payload type. The value for this field is in the range 96-127.
- PARMCH\_RFC2833\_EVT\_RX\_PLT - Identifies the receive payload type. The value for this field is in the range 96-127.

## 6.2.2 Setting In-Band Mode

In in-band mode, the DTMF audio is not clamped (not muted) and DTMF digits are sent in the RTP packets.

**Note:** When a channel is opened, the DTMF transfer mode is in-band by default.

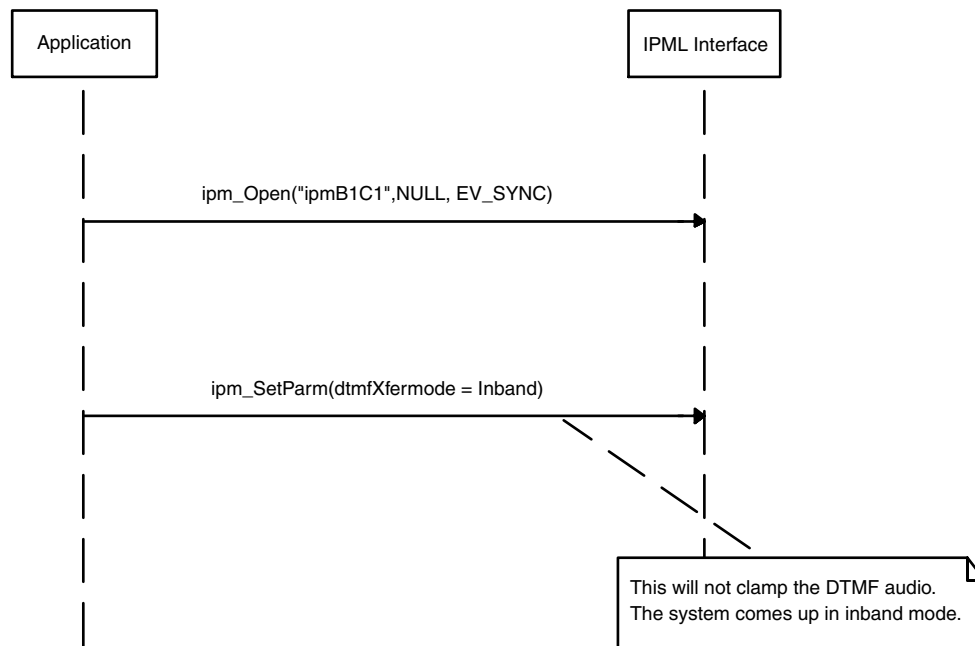
To set up a channel for in-band mode:

1. Open a channel using **ipm\_Open("ipmB1C1",NULL,EV\_SYNC)**
2. Set up the IPM\_PARM\_INFO structure and call **ipm\_SetParm()** as shown below:

```
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = DTMFXFERMODE_INBAND;
parmInfo.eParm = PARMCH_DTMFXFERMODE;
parmInfo.pvParmValue = &ulParmValue;
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
```

Figure 3 shows a scenario diagram for setting in-band mode.

Figure 3. In-Band Mode Scenario Diagram



### 6.2.3 Setting RFC 2833 Mode

To set up a channel for RFC 2833 mode, do the following:

1. Open a channel using **ipm\_Open("ipmB1C1",NULL,EV\_SYNC)**
2. Set the mode via the **IPM\_PARAM\_INFO** structure and **ipm\_SetParam()** as shown below:

```
IPM_PARAM_INFO paramInfo;
unsigned long ulParamValue = DTMFXFERMODE_RFC2833;
paramInfo.eParam = PARAMCH_DTMFXFERMODE;
paramInfo.pvParamValue = &ulParamValue;
ipm_SetParam(chdev, &paramInfo, EV_ASYNC)
```

3. Set up the RFC 2833 event payload on the transmit side as shown below:

```
IPM_PARAM_INFO paramInfo;
unsigned long ulParamValue = 101;
paramInfo.eParam = PARAMCH_RFC2833EVT_TX_PLT;
paramInfo.pvParamValue = &ulParamValue;
ipm_SetParam(chdev, &paramInfo, EV_ASYNC)
```

4. Set up the RFC 2833 event payload on the receive side as shown below:

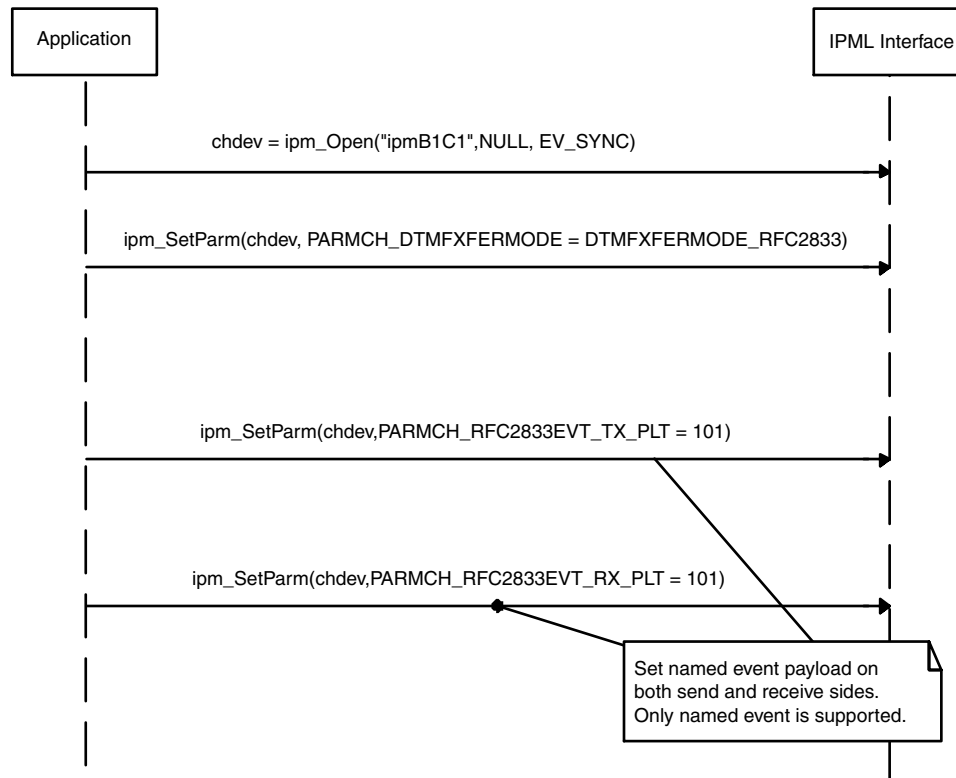
```
IPM_PARAM_INFO paramInfo;
unsigned long ulParamValue = 101;
paramInfo.eParam = PARAMCH_RFC2833EVT_RX_PLT;
paramInfo.pvParamValue = &ulParamValue;
ipm_SetParam(chdev, &paramInfo, EV_ASYNC)
```

5. Optionally, you can mute or un-mute the audio data in the RTP stream using the **eIPM\_RFC2833MUTE\_AUDIO** enumeration. The default state is for muting to be ON.

Figure 4 shows a scenario diagram for setting RFC 2833 mode.



Figure 4. RFC 2833 Scenario Diagram



## 6.2.4 Setting Out-of-Band Mode

In out-of-band mode, the DTMF audio is automatically clamped (muted) and DTMF digits are not sent in the RTP packets. To set up a channel for out-of-band mode, do the following:

1. Open a channel using **ipm\_Open("ipmB1C1", NULL, EV\_SYNC)**
2. Set the mode via the **IPM\_PARM\_INFO** structure and **ipm\_SetParm()** as shown below:

```

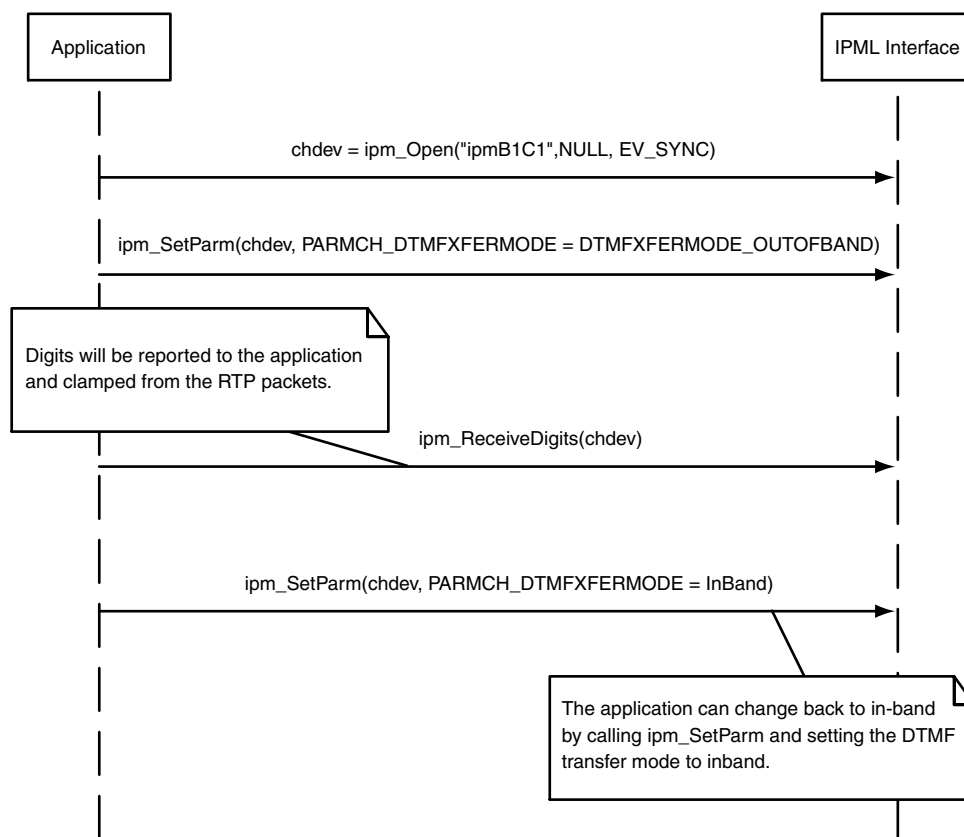
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = DTMFXFERMODE_OUTOFBAND;
parmInfo.eParm = PARMCH_DTMFXFERMODE;
parmInfo.pvParmValue = &ulParmValue;
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
    
```

3. Call **ipm\_ReceiveDigits(chdev)** to have digits reported to the application and clamped from the RTP packets.

To change back to in-band mode, set the **PARMCH\_DTMFXFERMODE** parameter to **DTMFXFERMODE\_INBAND**.

Figure 5 shows a scenario diagram for setting out-of-band mode.

Figure 5. Out-of-Band Mode Scenario Diagram



### 6.3 Notification of DTMF Detection

Notification of DTMF detection depends on the DTMF mode being used. For out-of-band mode, when an incoming DTMF digit is detected (received from the TDM bus), the application receives an unsolicited `IPMEV_DIGITS_RECEIVED` event. The event data is contained in `IPM_DIGIT_INFO`. One event is returned for each digit that is received.

For applications using Intel® NetStructure™ DM/IP Series boards and RFC 2833 mode, the application can request notification when DTMF digits are detected by using `ipm_EnableEvents()` with the `EVT_RFC2833` parameter. Once the events are enabled, when an incoming DTMF digit is detected, the application receives an unsolicited `IPMEV_RFC2833SIGNALRECEIVED` event. The event data is contained in `IPM_RFC2833_SIGNALID_INFO`.

## 6.4 Generating DTMF

Once DTMF mode has been configured, the application can generate DTMF digits using the `ipm_SendDigits()` function.

**Note:** The only supported direction for DTMF digit generation is towards the TDM bus.

Alternatively, the `ipm_SendRFC2833SignalIDToIP()` function can be used to send RFC 2833 data to the IP network.

**Note:** The `ipm_SendRFC2833SignalIDToIP()` function is not supported on Intel® NetStructure™ IPT Series boards. In this case, once you set the mode to RFC 2833, the only way to send an RFC 2833 digit is to have the `ipmBxCy` device listening to a TDM time slot. If the `ipmBxCy` device detects a digit from the TDM time slot, it will convert it to RFC 2833 and transmit the digit over RTP.

A typical use of the `ipm_SendRFC2833SignalIDToIP()` function is to:

- fill in the `IPM_RFC2833_SIGNALID_INFO` structure with the signal (tone) to send and the signal state set to `SIGNAL_STATE_ON` to start generating DTMF.
- call `ipm_SendRFC2833SignalIDToIP()` to indicate the start of the data
- wait an appropriate amount of time (for example, 50 msec)
- fill in the `IPM_RFC2833_SIGNALID_INFO` structure with the signal (tone) to stop and the signal state set to `SIGNAL_STATE_OFF` to stop generating DTMF.
- call `ipm_SendRFC2833SignalIDToIP()` to indicate the end of the data

This scenario is useful in situations when the application receives ringback from the PSTN and needs to send the tone data to the IP network. The application uses voice library functions to detect ringback. (See the *Voice API Library Reference* for more details.) Then the application sets the RFC2833 signal on and leave it on until the ringback stops.

## 6.5 Using T.38 Fax Gateway

The IP media software supports sending fax information during a session using the T.38 protocol, as shown in Figure 6.

**Note:** Another method of transferring fax information is to use the G.711 protocol. In this case, the fax data is sent from a fax-capable board in the system across the TDM bus. The IP media software can then send the data outside the system using IP. However, this method uses more bandwidth than the T.38 method.

To set up a channel to handle T.38 fax, do the following:

1. Open a channel using `ipm_Open("ipmB1C1",NULL,EV_SYNC)`
2. Enable event reporting using `ipm_EnableEvents(chDev, *pEvents)` and the `IPMEV_T38CALLSTATE` and `IPMEV_FAXTONE` events.
3. Get local RTP information using `ipm_GetLocalMediaInfo(chDev,&MediaInfo)` and setting the `eMediaType` field to `MEDIATYPE_LOCAL_RTP_INFO`
4. Start an RTP call using `ipm_StartMedia(chDev)`

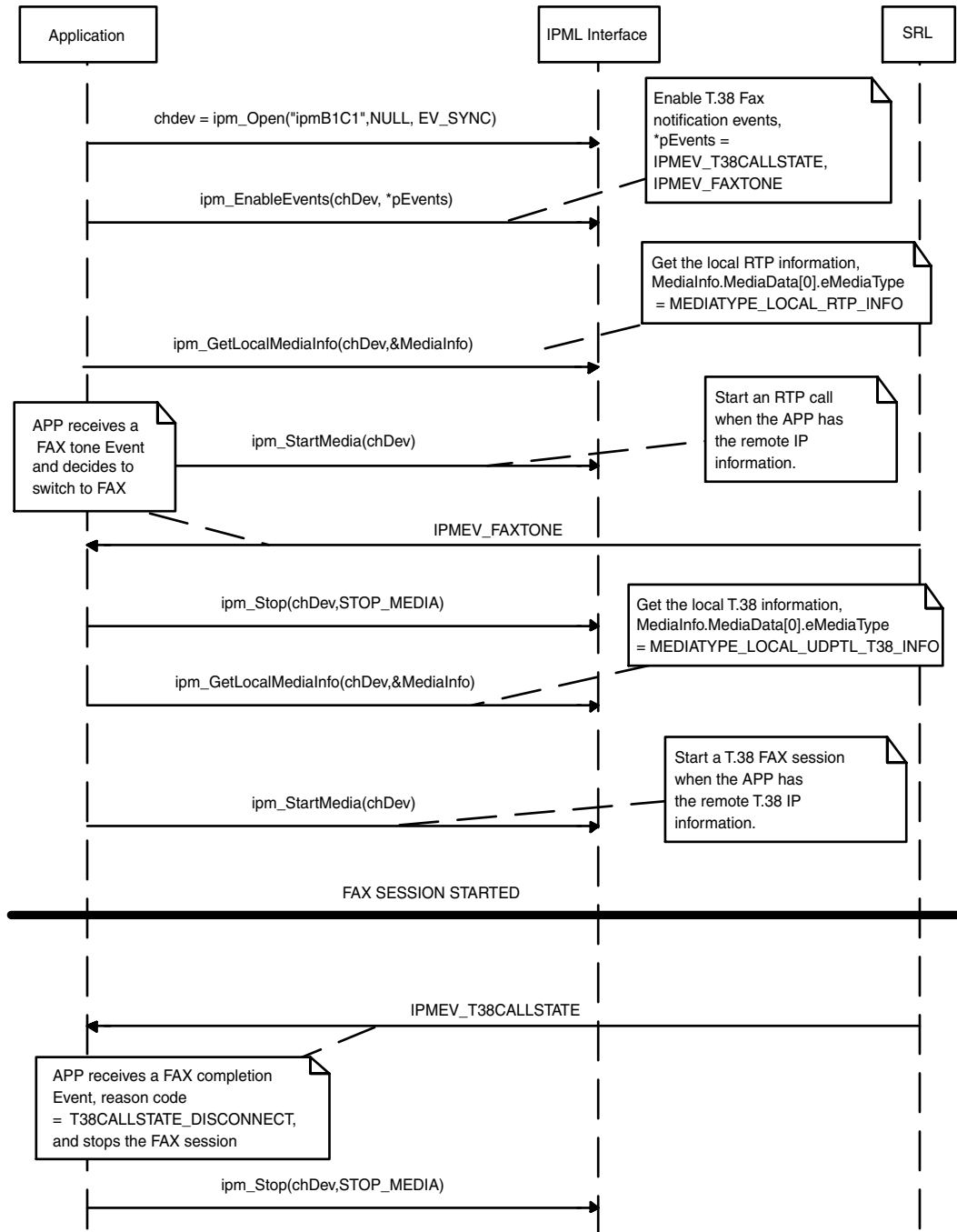
5. When the fax event IPMEV\_FAXTONE is received, the application should first stop the call in progress using ipm\_Stop(chDev,STOP\_MEDIA), then retrieve the local T.38 fax information using ipm\_GetLocalMediaInfo(chDev,&MediaInfo) and the eMediaType field MEDIATYPE\_LOCAL\_UDPTL\_T38\_INFO

**Note:** It is the responsibility of the application to respond promptly when the fax event is received or latency errors may occur. Refer to the T.38 Fax specification and ITU-T T.30 specification for latency guidelines. (Details on fax timing are in the T.30 specification.)

**Note:** CED and CNG tones must be exchanged before switching to T.38 mode. Also, CED and CNG tones will not transmit reliably over coders other than G.711 and G.726.

6. Once the remote fax information is available, the application then starts a fax session by calling ipm\_StartMedia(chDev)
7. When the fax event IPMEV\_T38CALLSTATE is received, with the reason code T38CALLSTATE\_DISCONNECT, the application can stop the fax session using ipm\_Stop(chDev,STOP\_MEDIA)

Figure 6. Using T.38 Fax Scenario Diagram





This chapter describes the QoS alarms that are supported by the IP media software, including the following sections:

- QoS Overview ..... 31
- QoS Alarm Types and Thresholds..... 31
- Alarm and Recovery Mechanisms..... 32
- Using QoS Alarms..... 34
- Hints for QoS Alarm Handling ..... 35

## 7.1 QoS Overview

The public switched telephone network (PSTN) defines quality of service as a particular level of service, for example “toll-like” service. However, quality of service for voice or other media over the Internet Protocol is defined as a continuum of levels, which are affected by packet delay or loss, line congestion, and hardware quality such as microphone quality. The IP media software is designed to operate along the entire range of quality of service, enabling the application to retrieve information necessary for correct billing.

All QoS parameters supported by the IP media software are disabled by default. That is, QoS monitoring must be enabled by the application. If desired, the application can set threshold values to monitor the quality of service during sessions. The QoS parameters are measured during time intervals, starting when a session is established. A fault occurs when the measurement of a QoS parameter crosses a predefined threshold. A success occurs when the measurement of a QoS parameter does not exceed a predefined threshold value.

## 7.2 QoS Alarm Types and Thresholds

All QoS alarms operate on a per-channel basis. That is, a QoS alarm indicates the status of particular channel during a particular session, not the status of an entire IP media resource board. The QoS alarms and thresholds described in this section are defined in the `IPM_QOS_THRESHOLD_DATA` data structure.

The following QoS alarms are identified by the enumeration `eIPM_QOS_TYPE`:

`EVT_DTMFDISCARDED`

number of lost DTMF digits since the beginning of the call; indicates the system could not process the digits that were sent (Intel® NetStructure™ DM/IP Series boards only)

`EVT_LOSTPACKETS`

percent of lost packets per second since the beginning of the call

**EVT\_JITTER**

average jitter since the beginning of the call (in msec)

**EVT\_ROUNDTRIP\_LATENCY**

RTP packet latency (Intel® NetStructure™ IPT Series boards only)

Each QoS alarm has the following threshold attributes:

*Note:* Only the **unFaultThreshold** field is supported on Intel® NetStructure™ IPT Series boards.

**unTime\_Interval**

amount of time between two QoS parameter measurements (in multiples of 100 msec)

**unDebounceOn**

time interval for measuring potential alarm set condition (in msec., must be multiple of **unTime\_Interval**).

**unDebounceOff**

time interval for measuring potential alarm clear condition (in msec., must be multiple of **unTime\_Interval**)

**unFaultThreshold**

alarm threshold value. When this value is exceeded, the **unDebounceOn** timer is triggered. The units for **unFaultThreshold** are dependent on what alarm is being measured. In the case of the Lost Packets alarm, **unFaultThreshold** is measured in lost packets per second.

**unPercentSuccessThreshold**

number of poll instances that the error threshold must be exceeded in a **unDebounce\_On** time interval before declaring alarm (expressed as a percentage of successes)

**unPercentFailThreshold**

number of poll instances that the error threshold was not exceeded in **unDebounce\_Off** time interval before clearing alarm (expressed as a percentage of failures)

## 7.3 Alarm and Recovery Mechanisms

Figure 7 shows how a QoS alarm is triggered and an event is sent to an application. The time line shows that QoS parameters are measured every **unTime\_Interval**. The QoS parameter in this example transitions from Success to Failure which starts the **unDebounceOn** timer. The QoS parameter transitions to Success and the **unDebounceOn** timer stops timing and is reset. At this time, the percentage failure rate is less than **unPercentFailThreshold** so no alarm event is sent. Later, the QoS parameter transitions from Success to Failure and the **unDebounceOn** timer begins timing again. In this scenario, the **unDebounceOn** timer expires while the QoS parameter is in Failure period and the percentage failure rate is equal to or greater than **unPercentFailThreshold**. Both of these situations cause an alarm event to be sent and the **unDebounceOn** timer to be reset.



Figure 7. Alarm Mechanism

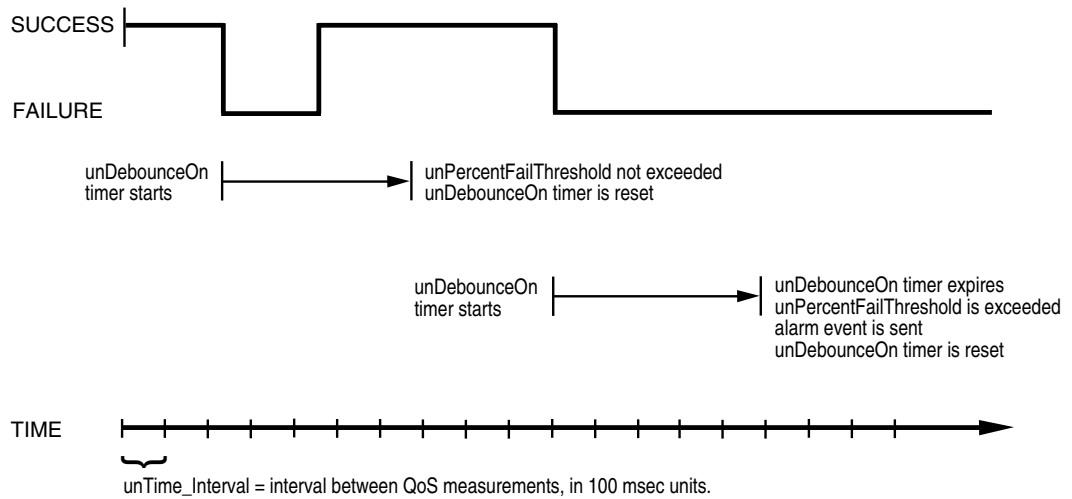
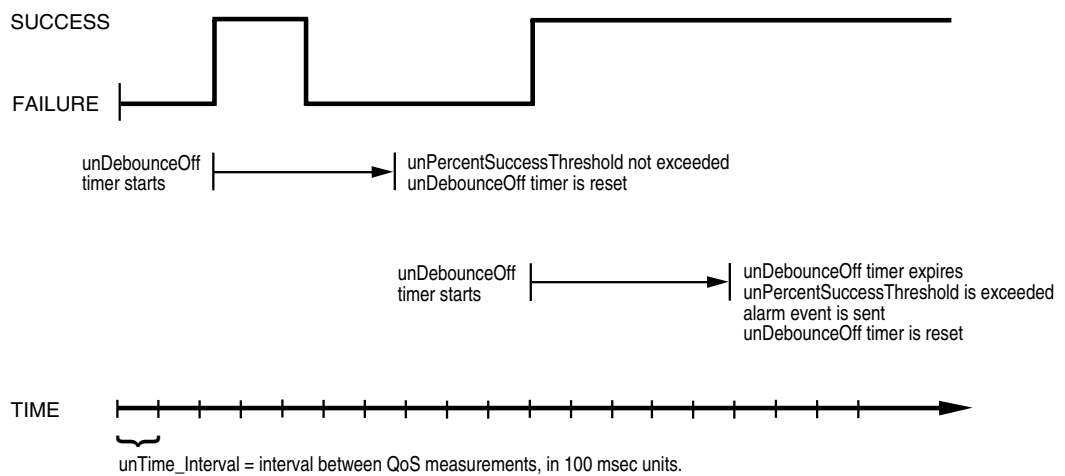


Figure 8 shows how an application recovers from a QoS alarm. The time line shows that QoS parameters are measured every **unTime\_Interval**. The QoS parameter in this example transitions from Failure to Success which starts the **unDebounceOff** timer. The QoS parameter transitions to Failure and the **unDebounceOff** timer stops timing and is reset. At this time, the percentage success rate is less than **unPercentSuccessThreshold** so no alarm recovery event is sent. Later, the QoS parameter transitions from Failure to Success and the **unDebounceOff** timer begins timing again. In this scenario, the **unDebounceOff** timer expires while the QoS parameter is in Success period and the percentage success rate is equal to or greater than **unPercentSuccessThreshold**. Both of these situations cause an alarm recovery event to be sent and the **unDebounceOff** timer to be reset.

Figure 8. Recovery Mechanism



## 7.4 Using QoS Alarms

The following steps are an overview of how to use QoS alarms in your application. For details, refer to the specific API and data structure descriptions:

1. Call **ipm\_GetQoSThreshold()** to retrieve the current settings of QoS parameters.
2. Set up the **IPM\_QOS\_THRESHOLD\_INFO** structure with desired values for QoS parameter settings.
3. Call **ipm\_SetQoSThreshold()** to set the desired QoS parameters. This function may be called at any time by an application, including while a session is in progress.
4. Call **ipm\_EnableEvents()** to start QoS parameter monitoring.
5. When a QoS alarm has been triggered, an **IPMEV\_QOS\_ALARM** event is sent by the system.
6. Use standard runtime library API functions to query the **IPMEV\_QOS\_ALARM\_DATA** structure to retrieve more details about the alarms which have been triggered.
7. Call **ipm\_DisableEvents()** to stop QoS parameter monitoring.

The following pseudocode illustrates how you might use QoS alarms in an application.

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

/* % PacketLoss */
Time_Interval= 1000;      /* 1 second          */
Debounce_On = 20000;     /* 20 seconds       */
Debounce_Off = 45000;    /* 45 seconds       */
Fault_Threshold = 6;     /* 2 lost 30 m sec packets per second.*/
Percent_Success_Threshold = 10; /* 10 seconds (in 20 seconds) */
Percent_Fail_Threshold = 30; /* 30 seconds (in 45 seconds) */

/* Jitter */
Time_Interval= 100;      /* 100 m sec        */
Debounce_On = 20000;     /* 20 seconds       */
Debounce_Off = 45000;    /* 45 seconds       */
Fault_Threshold = 360;    /* 1.5 30 m sec G711 packet jitter per second. (unscaled) */
Percent_Success_Threshold = 100; /* 10 seconds (in 20 seconds) */
Percent_Fail_Threshold = 300; /* 30 seconds (in 45 seconds) */

void main()
{
    int nDeviceHandle;
    eIPM_EVENT myEvents[] = {EVT_ROUNDTRIPLATENCY, EVT_LOSTPACKETS, EVT_JITTER};
    //Register eventhandler function with scl
    scl_enbhdr(EV_ANYDEV_EV_ANYEVT, (HDLR)CheckEvent);
    /*
    .
    Main Processing
    .
    */

    /*
    Need to enable three events for IP device handle, nDeviceHandle.
    ASSUMPTION : A valid nDeviceHandle was obtained from prior call to ipm_Open().
    */
    if (ipm_EnableEvents(nDeviceHandle, myEvents, 3, EV_SYNC) == -1)
    {
```

```

printf(" ipm_EnableEvents failed for device name %s with error = %d\n",
    ATDV_NAME_EP (nDeviceHandle), ATDV_LASTERR (nDeviceHandle));
/*
.
Perform Error Processing
.
*/
}
/*
.
Continue Processing
.
*/
}

void CheckEvent()
{
    int nEventType = sr_getevtype();
    int nDeviceID = sr_getevtdev();
    switch (nEventType)
    {
        /*
        .
        .List of expected events
        .
        */
        /* When alarm occurs you get this event. */

    case IPM_EV_QOS_ALARM :
        {
            printf("Received IPM_EV_QOS_ALARM for device = %s\n",
                ATDV_NAME_EP (nDeviceID));
            IPM_QOS_ALARM_DATA * l_pAlarm = (IPM_QOS_ALARM_DATA *) l_pVoid;
            switch (l_pAlarm->eQoSType)
            {
                case QOSTYPE_ROUNDTRIPLATENCY:
                    printf("QOSTYPE_ROUNDTRIPLATENCY=%d\n", l_pAlarm->eAlarmState);
                    break;
                case QOSTYPE_JITTER:
                    printf("QOSTYPE_JITTER=%d\n", l_pAlarm->eAlarmState);
                    break;
                case QOSTYPE_LOSTPACKETS:
                    printf("QOSTYPE_LOSTPACKETS=%d\n", l_pAlarm->eAlarmState);
                    break;
            }
            break;
        }

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

```

## 7.5 Hints for QoS Alarm Handling

The following hints may be useful in designing your application to handle QoS alarms:



If a QoS alarm occurs in the middle of a call and you wish to change coders, you must stop the particular media stream before you can change to a different coder. Note that stopping and restarting the media stream tears down the call, therefore, a brief interruption in the audio stream will likely be experienced.

For Intel® NetStructure™ IPT Series boards, the system software sends a QoS alarm event when a threshold is exceeded (ALARM\_STATE\_ON).

For Intel® NetStructure™ DM/IP Series boards and for HMP software, the system software sends a QoS alarm event when a threshold is exceeded (ALARM\_STATE\_ON) and when the threshold returns to the programmed level (ALARM\_STATE\_OFF).

This chapter contains the following sections:

- [Compiling and Linking under Linux](#) . . . . . 37
- [Compiling and Linking under Windows](#) . . . . . 38

## 8.1 Compiling and Linking under Linux

The following topics discuss compiling and linking requirements:

- [Include Files](#)
- [Required Libraries](#)

### 8.1.1 Include Files

To use IP media API functions in your Linux application, certain include files (also known as header files) and library files are required. You must add statements for these include files in your application. The following header files contain equates that are required for each Linux application that uses the IP media library:

*ipmerror.h*  
IP media library error header file

*ipmlib.h*  
IP media library header file

### 8.1.2 Required Libraries

The following library files must be linked to the application **in the following order**:

*libipm.so*  
Linking this file is mandatory. Specify `-lipm` in makefile.

*libgc.so*  
Required only if the application uses R4 Global Call library functions directly, for example, `gc_OpenEx()`. Specify `-lgc` in makefile.

*libdxxx.so*  
Required only if the application uses R4 voice library functions directly, for example, `dx_play()`. Specify `-ldxxx` in makefile.

*libsrl.so*  
Standard Runtime Library (SRL) is mandatory. Specify `-lsrl` in makefile.

*libpthread.so*  
POSIX threads system library. Specify `-lpthread` in makefile.

*libdl.so*

Dynamic Loader system library. Specify `-ldl` in makefile.

## 8.2 Compiling and Linking under Windows

The following topics discuss compiling and linking requirements:

- [Include Files](#)
- [Required Libraries](#)

### 8.2.1 Include Files

To use IP media library API functions in your Windows application, certain include files (also known as header files) and library files are required. You must add statements for these include files in your application. The following header files contain equates that are required for each Windows application that uses the IP media library:

*ipmerror.h*

IP media library error header file

*ipmlib.h*

IP media library header file

### 8.2.2 Required Libraries

The following library files must be linked to the application:

*libipm.lib*

Linking this file is mandatory.

*libgc.lib*

Required only if the application uses R4 Global Call library functions directly, for example, **gc\_OpenEx()**. Use the `-gc` argument to the system linker.

*libdxxxmt.lib*

Required only if the application uses R4 voice library functions directly, for example, **dx\_play()**.

*libsrlmt.lib*

Standard Runtime Library (SRL) is mandatory.



# Index

---

## A

- Alarm and Recovery Mechanisms 32
- Alarm Mechanism 33
- alarm mechanism 32
- alarm thresholds 31
- alarm types 31
- Application Development Guidelines 21
- Architecture 9
- ATDV\_ERRMSGP() 17, 19
- ATDV\_LASTERR() 17, 19
- ATDV\_NAMEP() 18
- ATDV\_SUBDEVS() 18

## B

- Building Applications 37

## C

- Compiling and Linking under Linux 37
- Compiling and Linking under Windows 38

## D

- dev\_Disconnect() 29
- DTMF
  - detection
    - notification of 26
  - generation 27
- DTMF mode 22
  - in-band 23
  - out-of-band 25
  - RFC 2833 24
- DTMF Modes 22
- dx\_play() 37, 38

## E

- eIPM\_QOS\_TYPE 31
- Error Handling 19
- Event Handling 17

## F

- Features 9
- figures
  - alarm mechanism 33
  - recovery mechanism 33

## G

- gc\_Listen() 21
- gc\_OpenEx() 37, 38
- Generating DTMF 27
- Global Call library
  - relationship with IP media 10

## H

- Hints for QoS Alarm Handling 35

## I

- in-band mode 23
- In-Band Mode Scenario Diagram 23
- Include Files 37, 38
- Introduction to DTMF Handling 21
- Introduction to the IP Media Library 10
- IP Media Architecture 10
- IP Media Channel State Diagram 15
- IP media software
  - Global Call relationship 10
  - media device naming 11
  - standard runtime library support 11
- ipm\_Close() 11
- IPM\_DIGIT\_INFO 26
- ipm\_DisableEvents() 34
- ipm\_EnableEvents() 22, 26, 34
- ipm\_GetQoSThreshold() 34
- ipm\_Listen() 21
- ipm\_Open 23, 24, 25
- ipm\_Open() 11
- IPM\_PARM\_INFO 22
- IPM\_QOS\_THRESHOLD\_DATA 31
- IPM\_QOS\_THRESHOLD\_INFO 34
- ipm\_ReceiveDigits 25

- ipm\_ReceiveDigits() 21
- IPM\_RFC2833\_SIGNALID\_INFO 26, 27
- ipm\_SendDigits() 22, 27
- ipm\_SendRFC2833SignalIDToIP() 27
- ipm\_SetParm() 22, 23, 24, 25
- ipm\_SetQoSThreshold() 34
- IPMEV\_QOS\_ALARM\_DATA 34

## M

- mechanisms for alarm and recovery 32
- Media Channel Device Naming 11
- media channel device naming 11
- mode
  - DTMF 22

## N

- Notification of DTMF Detection 26
- notification of DTMF detection 26

## O

- out-of-band mode 25
- Out-of-Band Mode Scenario Diagram 26

## P

- Product Description 9
- Programming Models 13

## Q

- QoS
  - alarm and recovery mechanisms 32
  - alarm thresholds 31
  - alarm types 31
  - overview 31
  - using alarms 34
- QoS Alarm Types and Thresholds 31
- QoS Overview 31
- quality of service 31
- Quality of Service (QoS) 31

## R

- Recovery Mechanism 33
- recovery mechanism 32
- Relationship with Global Call Library 10
- Required Libraries 37, 38

- RFC 2833 mode 24
- RFC 2833 Scenario Diagram 25

## S

- Setting DTMF Parameters 22
- Setting In-Band Mode 23
- Setting Out-of-Band Mode 25
- Setting RFC 2833 Mode 24
- sr\_dishdlr() 17
- sr\_enbhdlr() 17
- sr\_getboardcnt() 11
- sr\_getevtdev() 17
- sr\_getevttype() 17
- sr\_waitevt() 17
- sr\_waitevtEx() 17
- SRL Event Management Functions 17
- SRL Standard Attribute Functions 17
- Standard Runtime Library Support 11
- State Models 15

## T

- T.38 Fax Server Scenario 29
- thresholds for QoS alarms 31
- types of QoS alarms 31

## U

- Using QoS Alarms 34
- using QoS alarms 34
- Using the T.38 Fax Server 27