IP Media Library (IPML) API

Programming Guide

May 2006

05-1834-006

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

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

This IP Media Library (IPML) API 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 the express written consent of Intel Corporation.

Copyright © 2002-2006 Intel Corporation. All Rights Reserved.

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

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

Publication Date: May 2006

Document Number: 05-1834-006

Intel
1515 Route 10
Parsippany, NJ 07054

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

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

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

Contents

Figures

Tables

Revision History

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

Document No.	Publication Date	Description of Revisions
05-1834-006	May 2006	Configuring for Half- or Full-Duplex Media Streams chapter: New chapter.
05-1834-005	July 2005	Application Development Guidelines chapter: Replaced by two feature implementation chapters on DTMF Handling and Using T.38 Fax Gateway. DTMF Handling chapter: Removed references to unsupported IPM_RFC2833MUTE_AUDIO parameter (PTR #33826). Generating DTMF section: Removed information relating to unsupported ipm_SendRFC2833SignalIDToIP( ) function. Quality of Service (QoS) Alarms chapter: Heavily revised and reorganized. Removed DTMF discarded alarm type. Noted that lost packets alarm is only supported on IPT Series boards. Added new scenarios and graphics in QoS Alarm and Alarm Recovery Mechanisms section. Updated example code. Network Failure Alarm (IPT Series Boards Only): New section.
05-1834-004	November 2003	Published for System Release 6.0 on PCI for Windows. Application Development Guidelines chapter: Replaced HMP-specific T.38 Fax Server section with T.38 Fax Gateway information.
05-1834-003	September 2003	Published for Host Media Processing Software Release 1.1 for Windows. Application Development Guidelines chapter: Added HMP-specific T.38 Fax Server section in place of information on T.38 fax features not supported in HMP DTMF Modes: added value ranges for PARMCH_RFC2833_EVT_TX_PLT and PARMCH_RFC2833_EVT_RX_PLT
05-1834-002	November 2002	Application Development Guidelines chapter: Added Using T.38 Fax section
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 document provides programming guidelines for the IP Media Library API. It is a companion guide to the *IP Media Library API Library Reference*, which provides details on functions and parameters in the IP media software.

This document version is specific to the version of the IP media library that is provided in Intel® Dialogic® System Software releases that support the use of Intel NetStructure® DM/IP and Intel NetStructure® IPT board products.

Intended Audience

This guide is intended for software developers who will access and utilize 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.

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, "DTMF Handling", provides information on how to send and receive DTMF digits.
- Chapter 7, "Using T.38 Fax Gateway", provides information on implementing a T.38 fax gateway.
- Chapter 8, "Quality of Service (QoS) Alarms", details how QoS may be used in an application.
- Chapter 9, "Configuring for Half- or Full-Duplex Media Streams", describes how to configure endpoints for half- and full-duplex media streaming.
- Chapter 10, "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 Guide*
- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *Standard Runtime Library API Library Reference*
- Internet Engineering Task Force (IETF) Request for Comments RFC 2833, *RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals*, *http://www.ietf.org/rfc/rfc2833.txt*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/design/network/products/telecom/* (for product information)

Product Description 1

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

1.1 Features

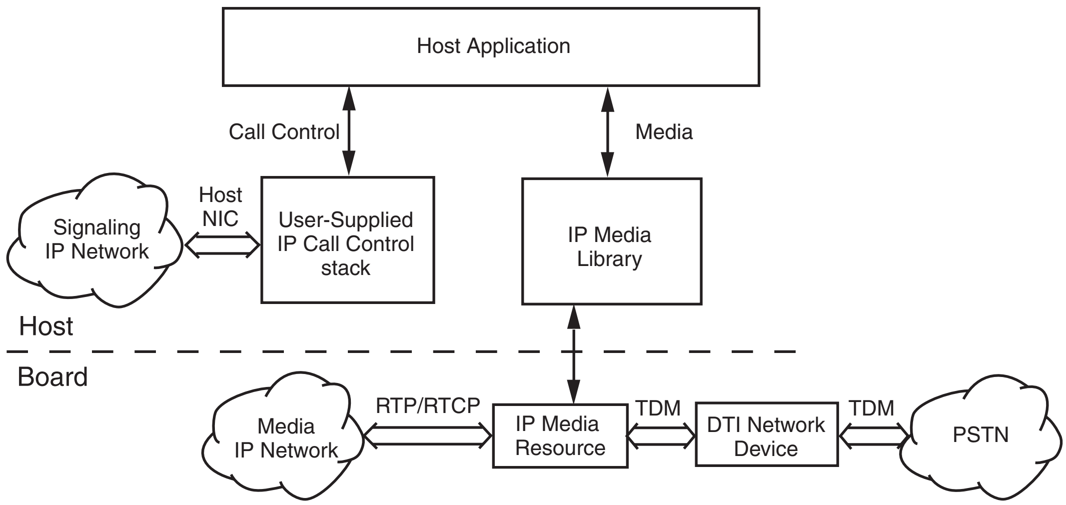
Some of the features of the IP media software include:

- 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 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.

Programming Models 2

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.

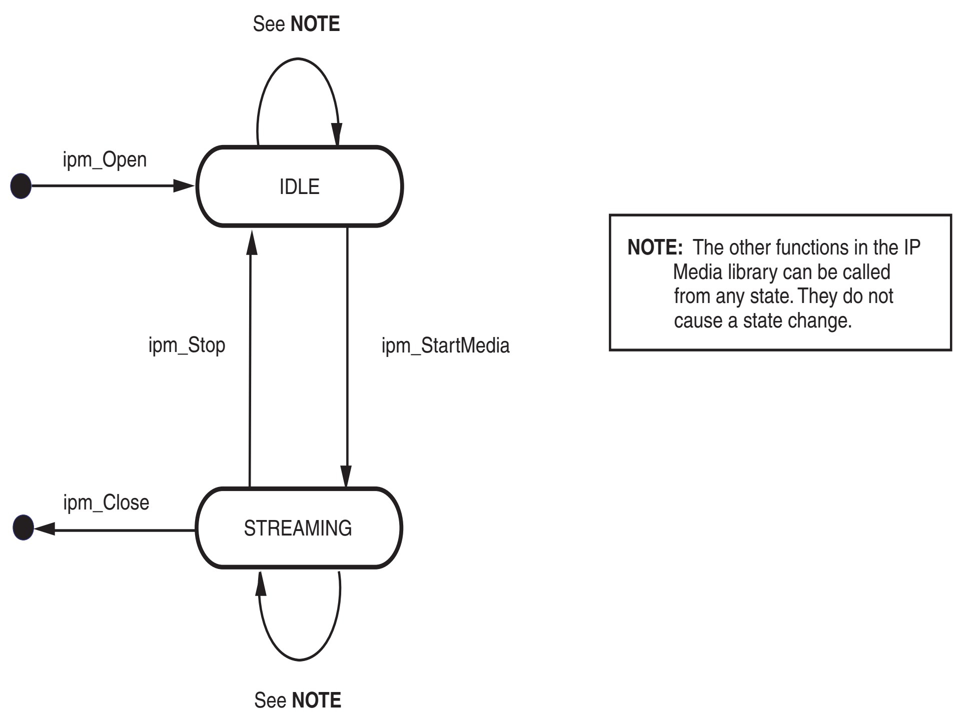
Note: The synchronous programming model is recommended for low density systems only. For high density systems, asynchronous programming models provide increased throughput for the application.

State Models 3

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



Event Handling 4

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.

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.

Error Handling 5

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( )**.

DTMF Handling 6

This chapter contains guidelines for implementing DTMF handling using the IP media library. The following topics are discussed:

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 12 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 in the RTP stream via RFC 2833 packets, 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 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.

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

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.

Note: For Intel NetStructure® DM/IP boards only, if you wish to be notified of RFC 2833 packets as they arrive at the IP port, your 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.

  Note: In-band mode cannot be used when using low bit-rate (LBR) coders.

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.

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 range for this field is 96 to 127.

PARMCH_RFC2833_EVT_RX_PLT
: Identifies the receive payload type. The value range for this field is 96 to 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 stream. When a channel is opened, the DTMF transfer mode is in-band by default.

Note: In-band mode cannot be used when using low bit-rate coders.

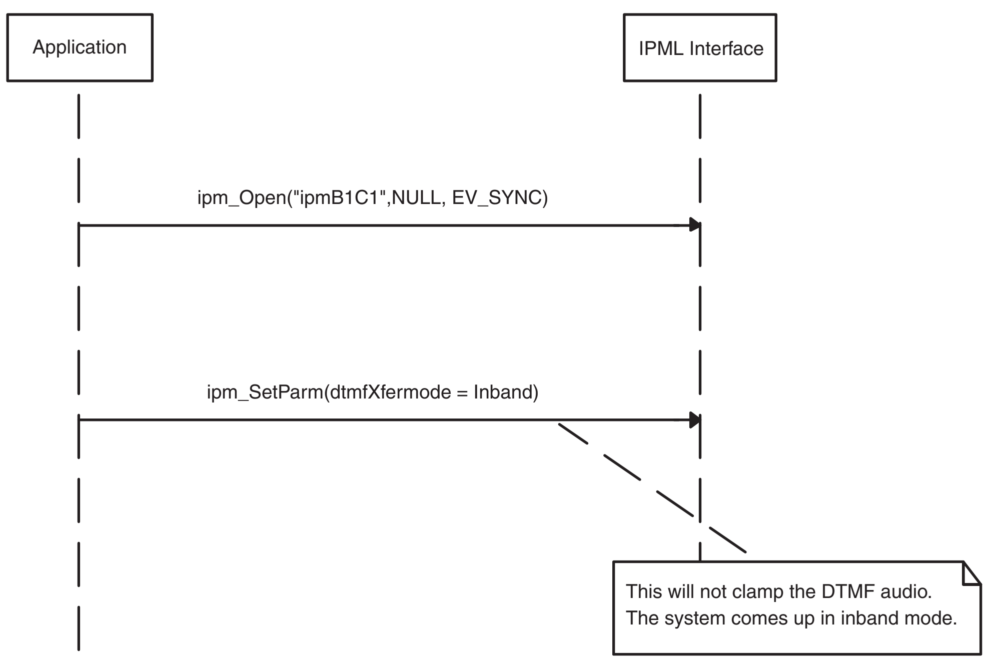
To set up a channel for in-band mode:

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 DTMF Mode Scenario Diagram



6.2.3 Setting RFC 2833 Mode

In RFC2833 mode, the DTMF audio is clamped (muted) and DTMF digits are sent in the RTP stream only as RFC2833 packets. 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_PARM_INFO structure and **ipm_SetParm**( ) as shown below:

```
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = DTMFXFERMODE_RFC2833;
parmInfo.eParm = PARMCH_DTMFXFERMODE;
parmInfo.pvParmValue = &ulParmValue
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
```

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

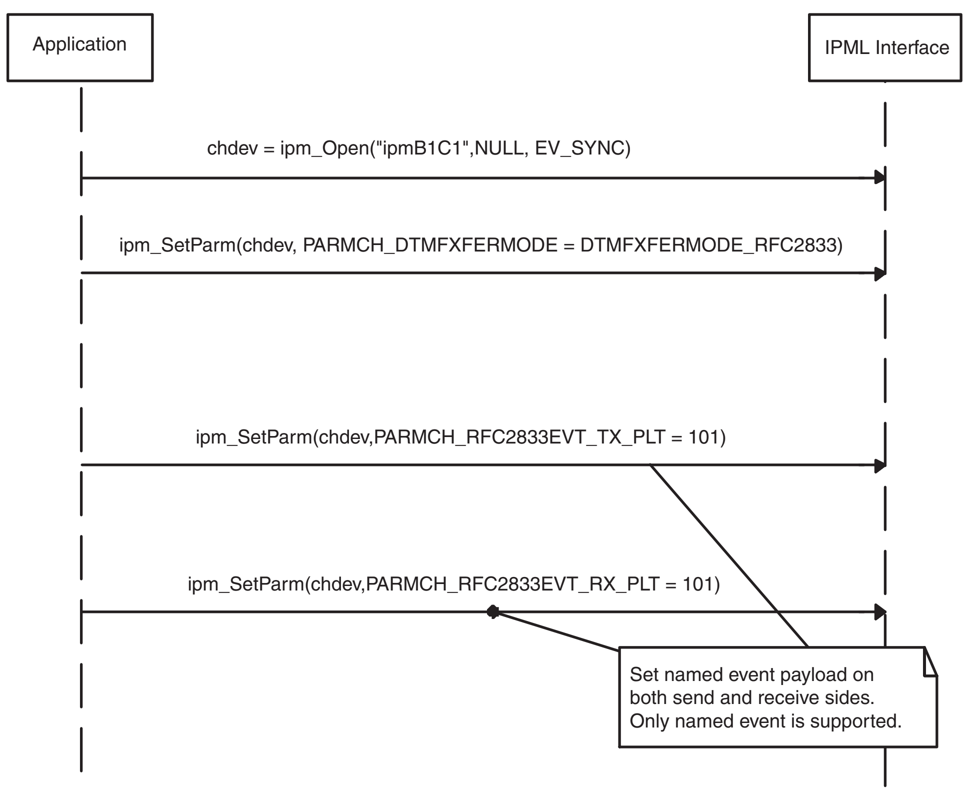
```
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = 101;
parmInfo.eParm = PARMCH_RFC2833EVT_TX_PLT;
parmInfo.pvParmValue = &ulParmValue
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
```

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

```
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = 101;
parmInfo.eParm = PARMCH_RFC2833EVT_RX_PLT;
parmInfo.pvParmValue = &ulParmValue
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
```

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:

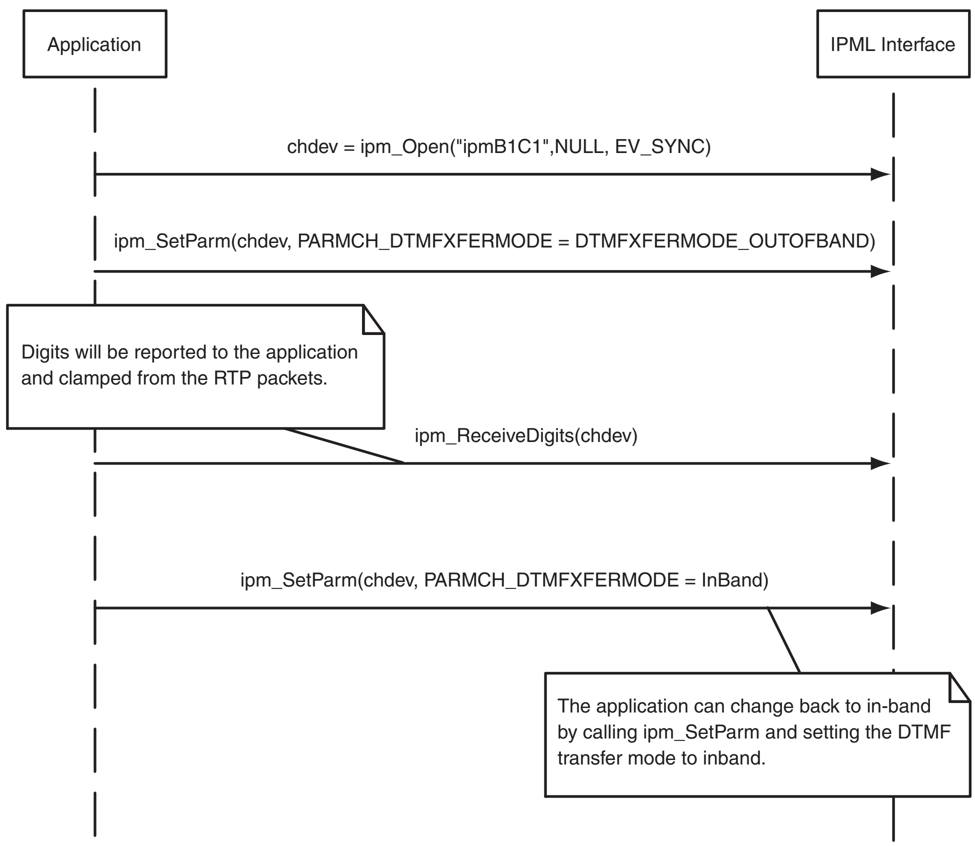
```
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = DTMFXFERMODE_OUTOFBAND;
parmInfo.eParm = PARMCH_DTMFXFERMODE;
parmInfo.pvParmValue = &ulParmValue
ipm_SetParm(chdev, &parmInfo, EV_ASYNC)
```

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 DTMF 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 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 IPT 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.

Using T.38 Fax Gateway 7

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

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.

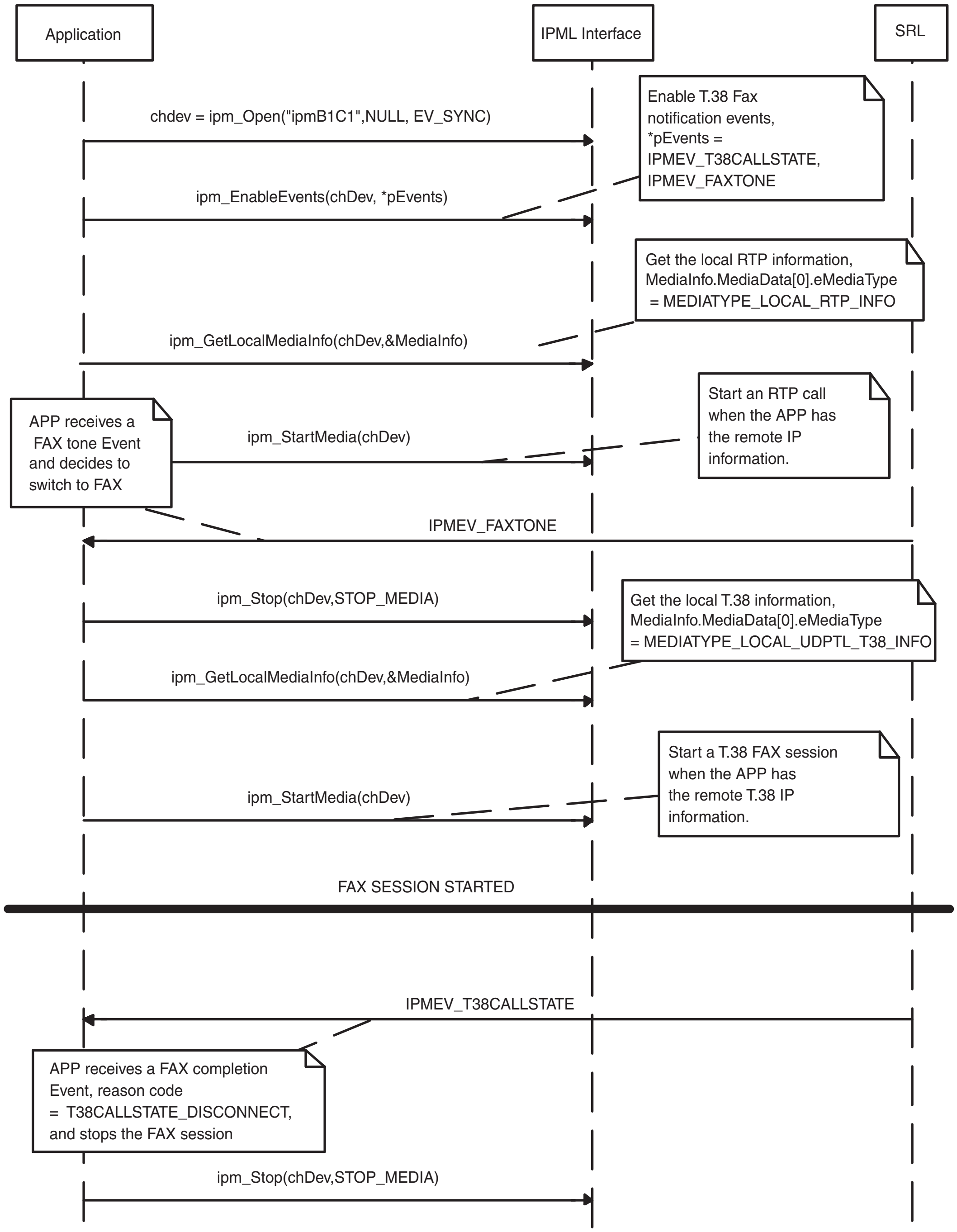
To set up a channel to handle T.38 fax, do the following:

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. T.38 Fax Scenario Diagram



Quality of Service (QoS) Alarms 8

This chapter describes the QoS alarms that are supported by the IP media software. The following topics are discussed:

8.1 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 exceeds a predefined threshold. A recovery occurs when the measurement of a QoS parameter returns to a value that does not exceed the predefined threshold.

To enable and use QoS monitoring in your application, you must follow several steps. Some steps are optional; others are required. These steps are detailed in Section 8.5, “Implementing QoS Alarms”, on page 34.

8.2 QoS Alarm Types

All QoS alarms operate on a per-channel basis. That is, a QoS alarm indicates the status of a particular channel during a particular session, not the status of an entire IP media resource board.

The following QoS alarm types are supported in the IP media software. These names are used in the IPM_QOS_THRESHOLD_DATA structure when setting parameters for the alarms, and in the IPM_QOS_ALARM_DATA structure that is associated with the IPMEV_QOS_ALARM event that is generated when an alarm state transition occurs.

QOSTYPE_JITTER
: QoS alarm for excessive average jitter

QOSTYPE_LOSTPACKETS
: QoS alarm for excessive percentage of lost packets

  Note: This alarm is supported on Intel NetStructure IPT boards only.

QOSTYPE_ROUNDTRIPLATENCY
: QoS alarm for excessive RTP packet latency

  Note: This alarm is supported on IPT boards only.

Note: Intel NetStructure IPT boards also support a board-level network failure alarm whose name shares the "QOSTYPE_" prefix even though the alarm is not a true QoS alarm. The network failure alarm is discussed in Section 8.8, "Network Failure Alarm (IPT Boards Only)", on page 42.

For details on using QoS alarms in your application, see Section 8.5, "Implementing QoS Alarms", on page 34.

8.3 QoS Threshold Attributes

All QoS alarm types have one or more threshold attributes, such as time interval and fault threshold, which specify how the system determines when to generate a QoS alarm event.

The threshold attributes listed below are specified in IPM_QOS_THRESHOLD_DATA structures that are contained in an IPM_QOS_THRESHOLD_INFO structure that is passed to **ipm_SetQoSThreshold( )**:

unTimeInterval
: time interval between successive parameter measurements

unDebounceOn
: polling interval for detecting potential alarm fault condition. This interval must be a multiple of unTimeThreshold.

unDebounceOff
: polling interval for measuring potential alarm non-fault condition. This interval must be a multiple of unTimeThreshold.

unFaultThreshold
: fault threshold value. The meaning and value range of this attribute depend on the alarm type.

unPercentSuccessThreshold
: percentage of poll instances in unDebounceOff interval that the fault threshold must not be exceeded before an "alarm off" event is sent. The granularity for this attribute is the ratio of unTimeInterval to unDebounceOff, expressed as a percentage.

unPercentFailThreshold
: percentage of poll instances in unDebounceOn interval that the fault threshold must be exceeded before an "alarm on" event is set. The granularity for this attribute is the ratio of unTimeInterval to unDebounceOff, expressed as a percentage.

Note: Not all attributes are supported for all alarm types and products. For example, on Intel NetStructure IPT boards, the only supported attribute is unFaultThreshold. All attributes that are not supported should be set to 0.

The IP Media Library software provides default values for each threshold attribute that will be used if the application does not specify any threshold values via **ipm_SetQoSThreshold( )**; the specific default values vary by product and by alarm type. The following table provides details on the attributes supported and the default values for each QoS alarm type. Note that if the application needs to set an explicit value for any of the threshold values for a particular alarm, it must specify the values for *all* fields in the IPM_QOS_THRESHOLD_DATA structure, including those that remain at default values.

For details on the IPM_QOS_THRESHOLD_DATA structure, see the *IP Media Library API Library Reference*.

Table 1. Quality of Service Parameter Defaults for IPT Boards

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold [1]	% Success Threshold	% Fail Threshold
Jitter	0	0	0	30 (ms)	0	0
Lost Packets	0	0	0	100 (%)	0	0
Round-trip Latency	0	0	0	950 (ms)	0	0

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

Table 2. Quality of Service Parameter Defaults for DM/IP Boards

QoS Type	Time Interval (ms)	Debounce On (ms)	Debounce Off (ms)	Fault Threshold [1]	% Success Threshold	% Fail Threshold
Jitter	5000	20000	60000	60 (ms)	40	40

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

8.4 QoS Events

The following QoS event types are used when calling the **ipm_EnableEvents( )** and **ipm_DisableEvents( )** functions to enable and disable the corresponding QoS alarms.

EVT_JITTER
: event indicating excessive jitter

EVT_LOSTPACKETS
: event indicating excessive percentage of lost packets (supported on Intel NetStructure IPT boards only)

EVT_ROUNDTRIPLATENCY
: event indicating excessive RTP packet latency (supported on Intel NetStructure IPT boards only)

These QoS events correspond to the QoS alarms discussed in Section 8.2, "QoS Alarm Types", on page 31. For details on enabling QoS alarms in your application, see the following section, "Implementing QoS Alarms".

8.5 Implementing QoS Alarms

The following steps provide general guidelines for implementing QoS alarms in your application. For details on the IP Media Library functions and data structures that are mentioned, see the *IP Media Library API Library Reference*.

Note: These steps do not represent every task that must be performed to create a working application but are intended as general guidelines.

1. **Optional steps before enabling a QoS alarm:**
   a. Call **ipm_GetQoSThreshold( )** to retrieve the current settings of QoS parameters on the specified IP channel. QoS parameter default values vary by alarm type and product. For information on QoS parameter default values, see the table in Section 8.3, "QoS Threshold Attributes", on page 32.
   b. If you need to change current QoS parameter values, set up the IPM_QOS_THRESHOLD_INFO structure with desired values. This structure contains one or more IPM_QOS_THRESHOLD_DATA structures. Note that you must explicitly specify the value for *every* parameter in the IPM_QOS_THRESHOLD_DATA structure, even if you want to use the default value for some of those parameters and non-default values for other parameters.
   c. Call **ipm_SetQoSThreshold( )** to use the QoS parameter values set in step 1b.
2. **Enable QoS alarms and start media streaming:**
   a. Call **ipm_EnableEvents( )** to enable QoS monitoring for a list of alarm types.
   b. Call **ipm_StartMedia( )** to start media streaming and begin QoS monitoring.
3. **Monitor QoS alarm notification events:**
   a. When a QoS alarm has been triggered, an IPMEV_QOS_ALARM event is generated by the system. Call the Standard Runtime Library function **sr_getevttype( )** to return the event type.
   b. Use Standard Runtime Library API functions such as **sr_getevtdatap( )** to query the IPM_QOS_ALARM_DATA structure to learn whether the alarm state is on or off.

   Note: For DM/IP boards, the system sends a QoS alarm event containing ALARM_STATE_ON when the fault threshold is exceeded and sends a QoS alarm event containing ALARM_STATE_OFF when the threshold returns to the programmed level. For IPT boards, the system software sends a QoS alarm event containing ALARM_STATE_ON only when a fault threshold is exceeded, but does **not** report a QoS event when the threshold returns to the programmed level.

4. **Perform clean-up activities:**
   a. Call **ipm_Stop( )** to stop media streaming.
   b. Call **ipm_DisableEvents( )** to stop QoS parameter monitoring.

For example code that illustrates how to implement QoS alarms, see Section 8.7, "Example Code for QoS Alarm Handling", on page 39.

8.6 QoS Alarm and Alarm Recovery Mechanisms

The information in this section only applies to DM/IP boards. IPT boards do not support the debounce parameters and do not generate "alarm off" events when a channel has recovered from an alarm condition.

To explain how the system monitors, detects, and clears a QoS alarm condition, three scenarios will be presented. In the first scenario, a QoS fault condition is detected but an alarm-on event is not sent to the application. In the second scenario, the QoS fault condition meets all alarm criteria and an alarm-on event is sent. The third scenario expands on the second scenario and describes how the alarm-on condition is cleared.

These scenarios are intended to illustrate the concepts. For easier reference, in the figures, time is shown in seconds rather than in millisecond units. For details on the parameters, see the *IP Media Library API Library Reference*.

In the three scenarios, the jitter alarm type is being monitored. The QoS parameters (alarm threshold attribute values) used in these scenarios are:

- unTimeInterval = 1000 ms (1 second)
- unDebounceOn = 4000 ms (4 seconds)
- unDebounceOff = 4000 ms (4 seconds)
- unFaultThreshold = 60 milliseconds
- unPercentFailThreshold = 50 percent
- unPercentSuccessThreshold = 50 percent

From these parameters, the library calculates "count" values for alarm-on and alarm-off debouncing that represent the number of measurements that must fail (or succeed) within a unTimeInterval period before an alarm-on (or alarm-off) event is generated.

For alarm-on debouncing:
count = int((**unDebounceOn/unTimeInterval**) * (**unPercentFailThreshold**/100))
= int((4000/1000) * (50/100))
= int(4 * 0.5)
= 2

For alarm-off debouncing:

count = int((**unDebounceOff/unTimeInterval**) * (**unPercentSuccessThreshold/**100))
= int((4000/1000) * (50/100))
= int(4 * 0.5)
= 2

For example code that uses these QoS parameter values, see Section 8.7, "Example Code for QoS Alarm Handling", on page 39.

Scenario 1: Brief Alarm Condition

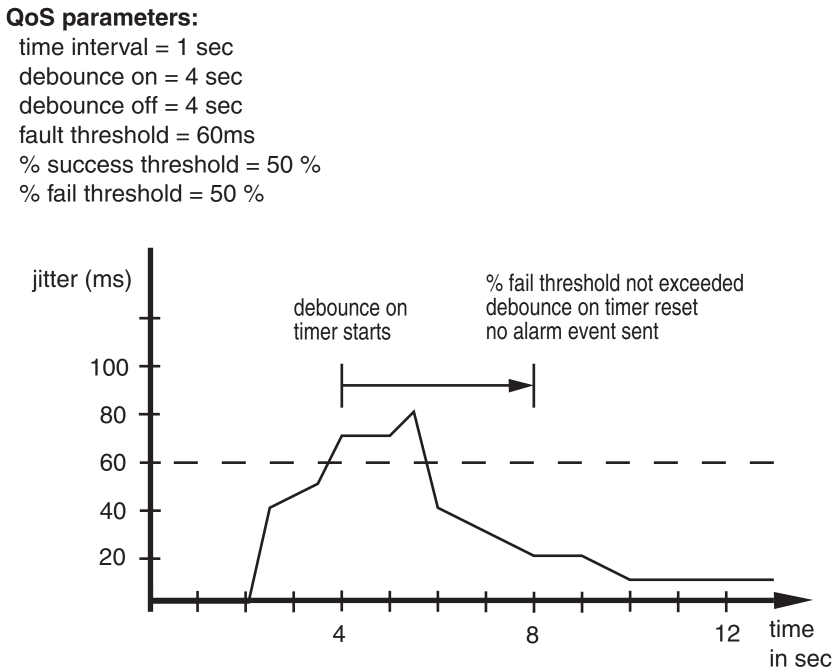
This scenario illustrates that a QoS alarm is triggered, but the alarm condition does not meet all of the specified alarm criteria. An alarm-on event is not sent to the application.

In Figure 7, the time line shows that QoS parameters are measured every time interval (**unTimeInterval** parameter), or every 1 second in this case. When the jitter exceeds the 60ms fault threshold (**unFaultThreshold** parameter), the debounce on timer is kicked off (**unDebounceOn** parameter). In this example, the fault threshold is exceeded at the 4th second.

To determine if this is a true alarm condition, the system continues to monitor the jitter in blocks of 4 seconds (**unDebounceOn** parameter), the debounce on window. If the jitter is below the 60ms fault threshold for more than 50 percent of the time (**unPercentFailThreshold** parameter) in a 4-second block, an alarm-on event is not sent to the application.

In this example, at the end of the 4-second debounce on window (at the 8th second), the percent failure threshold measured is 25 percent; that is, the fault threshold only exceeded the desired fault threshold of 60ms at the 5th second measurement within the 4-second debounce on window. Since the desired percentage failure threshold of 50 percent was not met or exceeded, no alarm-on event is sent to the application. At the end of the 8th second, the debounce on timer is reset.

Figure 7. QoS Scenario 1: Brief Alarm Condition



Scenario 2: True Alarm Condition

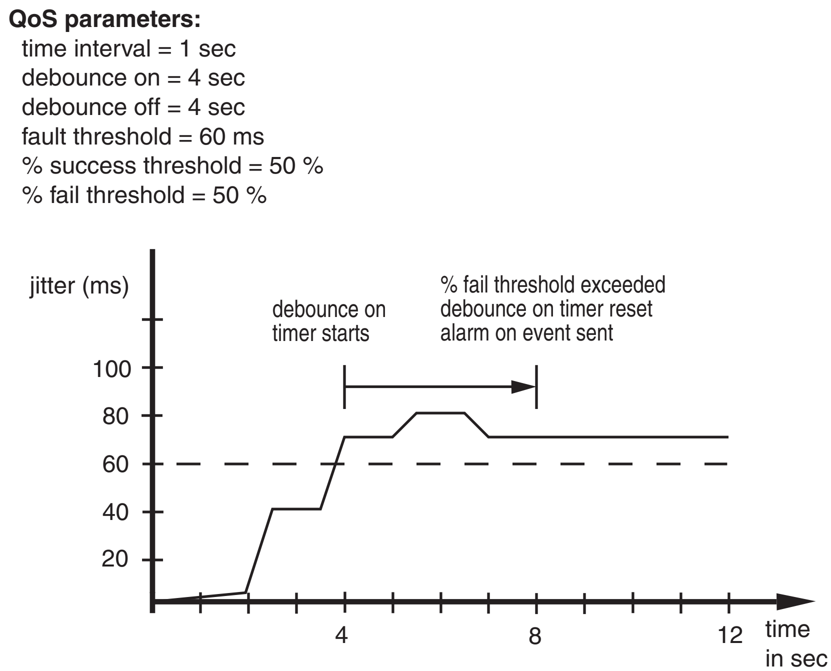
This scenario illustrates that a QoS alarm is triggered, and the alarm condition meets all of the specified alarm criteria. Therefore, an alarm-on event is sent to the application.

In Figure 8, the time line shows that QoS parameters are measured every time interval (**unTimeInterval** parameter), or every 1 second in this case. When the jitter exceeds the 60ms fault threshold (**unFaultThreshold** parameter), the debounce on timer is kicked off (**unDebounceOn** parameter). In this example, the fault threshold is exceeded at the 4th second.

To determine if this is a true alarm condition, the system continues to monitor the jitter in blocks of 4 seconds, the debounce on window (**unDebounceOn** parameter). If the jitter exceeds the 60ms fault threshold for more than 50 percent of the time (**unPercentFailThreshold** parameter) in a 4-second block, an alarm-on event is sent to the application.

In this example, at the end of the 4-second debounce on window (at the 8th second), the percent failure threshold measured is 100 percent; that is, the fault threshold exceeded the desired fault threshold of 60ms at the 5th, 6th, 7th and 8th second measurement within the 4-second debounce on window. Since the desired percentage failure threshold of 50 percent was exceeded, an alarm-on event is sent to the application. At the end of the 8th second, the debounce on timer is reset. See Scenario 3: Alarm Condition Cleared to learn how the system continues to monitor the jitter QoS alarm.

Figure 8. QoS Scenario 2: True Alarm Condition



Scenario 3: Alarm Condition Cleared

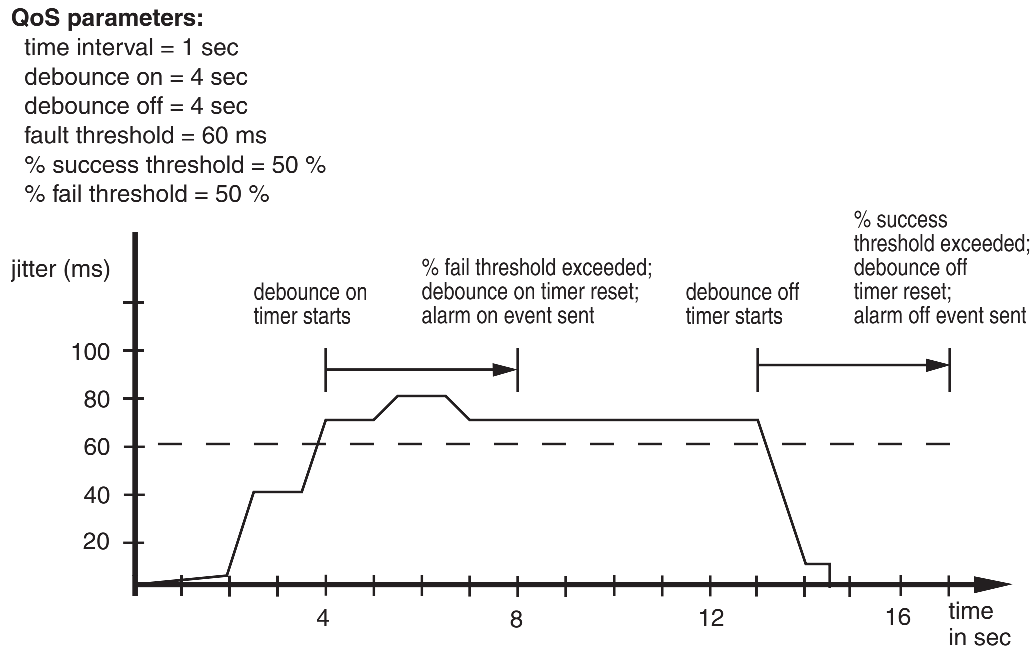
Scenario 3 builds on Scenario 2 to illustrate what happens after an alarm-on event is sent to the application and how the alarm-on condition is cleared.

In Figure 9, an alarm-on event was sent to the application at the 8th second, and the system is now in a QoS failure condition. To determine how long this condition will last, the system resumes monitoring the jitter every time interval (**unTimeInterval** parameter), or every 1 second in this case. When the jitter is less than the 60ms fault threshold (**unFaultThreshold** parameter), the debounce off timer kicks in (**unDebounceOff** parameter). In this example, this condition occurs at the13th second.

To determine if this is a true success condition, the system monitors the jitter in blocks of 4 seconds, the debounce off window (**unDebounceOff** parameter). If the jitter is below the 60ms fault threshold for more than 50 percent of the time (**unPercentSuccessThreshold** parameter) in a 4-second block, an alarm-off event is sent to the application.

In this example, at the end of the 4-second debounce off window (at the 17th second), the percent success threshold measured is 100 percent; that is, the jitter level was below the desired fault threshold of 60ms at the 14th through 17th second measurement within the 4-second debounce off window. Since the desired percentage success threshold of 50 percent was exceeded, an alarm-off event is sent to the application. At the end of the 17th second, the debounce off timer is reset.

Figure 9. QoS Scenario 3: Alarm Condition Cleared



8.7 Example Code for QoS Alarm Handling

The following pseudocode illustrates how you might use QoS alarms in an application. The code enables the following QoS alarm types: jitter, lost packets, RTCP timeout, and RTP timeoutjitter and lost packets (which is only supported on IPT boards). Because the IPM_QOS_THRESHOLD_INFO structure is not filled in for the lost packets alarm type, the default QoS parameter values are used for this alarm. The QoS parameter values for jitter are the same values used in the scenario descriptions in Section 8.6, "QoS Alarm and Alarm Recovery Mechanisms", on page 35.

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

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


void main()
{
   int nDeviceHandle;
   IPM_QOS_THRESHOLD_INFO mySetQosThresholdInfo;

   const int nNumEvent = 2;
   eIPM_EVENT myEvents[nNumEvent] ={EVT_LOSTPACKETS, EVT_JITTER;
```

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


/*
.
.
    The application can call ipm_GetQoSThreshold() to check the current
    threshold levels for QoS parameters.
.
.
*/


// Change alarm threshold settings for IP device handle, nDeviceHandle.
// ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
// Note:
// 1. You don't have to change all QoS types. In the example below, the lost packet
//    values are not changed.
mySetQosThresholdInfo.unCount = 3;
mySetQosThresholdInfo.QoSThresholdData[0].eQoSType = QOSTYPE_JITTER;
mySetQosThresholdInfo.QoSThresholdData[0].unTimeInterval = 1000; //1sec
mySetQosThresholdInfo.QoSThresholdData[0].unDebounceOn = 4000; //4sec
mySetQosThresholdInfo.QoSThresholdData[0].unDebounceOff = 4000;//4sec
mySetQosThresholdInfo.QoSThresholdData[0].unFaultThreshold = 60;//60ms
mySetQosThresholdInfo.QoSThresholdData[0].unPercentSuccessThreshold = 50;//50%
mySetQosThresholdInfo.QoSThresholdData[0].unPercentFailThreshold = 50;//50%

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


// Call ipm_EnableEvent to be notified of possible alarm conditions.

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

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

    // Appplication can disable events if it does not want to be notified.

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

    if(ipm_Close(nDeviceHandle, NULL) == -1)
    {
        printf("----------->ipm_Close() failed for handle = %d\n", nDeviceHandle);
        /*
        .
        .
        Perform Error Processing
        .
        .
        */
    }
}


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

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

    /* When alarm occurs you get this event. */
    case IPMEV_QOS_ALARM:
        {
            printf("Received IPMEV_QOS_ALARM for device = %s\n",
                ATDV_NAMEP(nDeviceID));
            IPM_QOS_ALARM_DATA * l_pAlarm = (IPM_QOS_ALARM_DATA*)pVoid;
            switch(l_pAlarm->eQoSType)
            {
            case QOSTYPE_JITTER:
                printf("Alarm Type = Jitter\n");
                break;
            case QOSTYPE_LOSTPACKETS:
                printf("Alarm Type = LostPackets\n");
                break;
            }
            printf("Alarm state = %s\n", (l_pAlarm->eAlarmState? "On": "Off"));
            break;
        }
```

```
        /*
        .
        .
            process other cases.
        .
        .
        */

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

8.8 Network Failure Alarm (IPT Boards Only)

Intel NetStructure IPT boards support a board-level alarm to notify the application when the board's connection to the RTP network has been disrupted, for example if the Ethernet cable has been disconnected or if there has been some failure in a hub or switch. When the alarm is enabled, the board checks the status of the network connection at 1 second intervals. If the board finds that the connection is disrupted, it generates a single network failure event to notify the application. When a subsequent network status check indicates that the network connection has been restored, a single network failure alarm off event is generated. Both the alarm-on (failure) and alarm-off (restoration) events may also be reported to the Global Call library via the GCAMS mechanism.

The network failure alarm uses much the same programming interface as QoS alarms, but there are some significant differences between the two types of alarms:

- The network failure alarm is a board-level alarm while the QoS alarms operate at the channel device level.
- The network failure alarm generates an event when the loss of the network connection is detected and another when the connection is restored. QoS alarms on IPT boards only generate an event when the threshold for the alarm condition is first exceeded; no event is generated when the QoS returns to the acceptable range.
- There are no threshold parameters associated with the network failure alarm.
- The network failure alarm cannot be reset via **ipm_ResetQoSAlarmStatus( )**.
- The status of the network failure alarm cannot be queried via **ipm_GetQoSAlarmStatus( )**, and the status is not reported via **ipm_GetSessionInfo( )**. The network failure alarm is only reported via asynchronous notification events.

The network failure alarm is only available on an IPT board that has been explicitly opened by the application. The alarm is not available when the board is opened implicitly or automatically.

The application registers for notification of the network failure alarm in much the same way as a QoS alarm, by calling **ipm_EnableEvent( )**, and deregisters via **ipm_DisableEvent( )**. It is important to note that a separate function call must be used to enable or disable the network failure alarm event, because the device handle that is passed to the function must be a board device handle when setting the EVT_NETWORKFAILURE event type. The function call will fail if a channel device handle is specified.

Event handling in IPML for the network failure alarm is identical to that for QoS alarm events, except that the handler needs to distinguish between alarm-on and alarm-off events. The event that is generated when a network failure is detected is of type IPEV_QOS_ALARM, and it contains associated data of type IPM_QOS_ALARM_DATA. The eQoSType field of this data structure is QOSTYPE_NETWORKFAILURE, and the eAlarmState may be either ALARM_STATE_ON or ALARM_STATE_OFF.

Configuring for Half- or Full-Duplex Media Streams 9

This chapter describes the IP Media Library capability for setting up and reconfiguring end points for half-duplex and full-duplex media streams.

9.1 Overview of Half- and Full-Duplex Stream Support

When using IP technology, the ability to start a stream session at an end point for half-duplex (or full-duplex) media streaming and subsequently update the stream session for full-duplex (or half-duplex) media streaming is a useful capability.

One example that demonstrates this is when providing support for a call control feature known as “Early Media”. In IP technology, the establishment of RTP media streaming is normally one of the final steps in establishing and connecting a call. This is in contrast to the Public Switched Telephone Network (PSTN), where call progress signaling is commonly provided to the calling party via audible, in-band call progress tones, such as ringback, busy signal, and SIT tones. When implementing a VoIP gateway, it is often imperative to initiate media (RTP) streaming from the local endpoint to the calling party before the call is connected.

To achieve this functionality using the IP Media Library, the calling party end point can be configured for half-duplex streaming (in the receive direction) prior to call connection to receive call progress signaling. The end point can then be reconfigured for full-duplex streaming when the call is connected.

Another useful application of this feature is in the implementation of H.450.4 call hold message flows.

Note: Configuring half-duplex streaming for T.38 fax does not apply, since T.38 fax in inherently full-duplex.

9.2 Programming Interface for Half- and Full-Duplex Stream Configuration

The following IP Media Library functions are used to configure and reconfigure an end point for half-duplex or full-duplex media streaming:

ipm_StartMedia( )
sets media properties and starts an RTP media streaming session

ipm_ModifyMedia( )
modifies various properties of an active RTP media streaming session

Both functions contain an **eDirection** parameter that is used to specify if the media stream should be half-duplex or full-duplex. Possible values of the **eDirection** parameter are:

DATA_IP_RECEIVEONLY
receive RTP and RTCP packets from the IP network, but do not send packets

DATA_IP_SENDONLY
send RTP and RTCP packets to the IP network, but do not receive packets

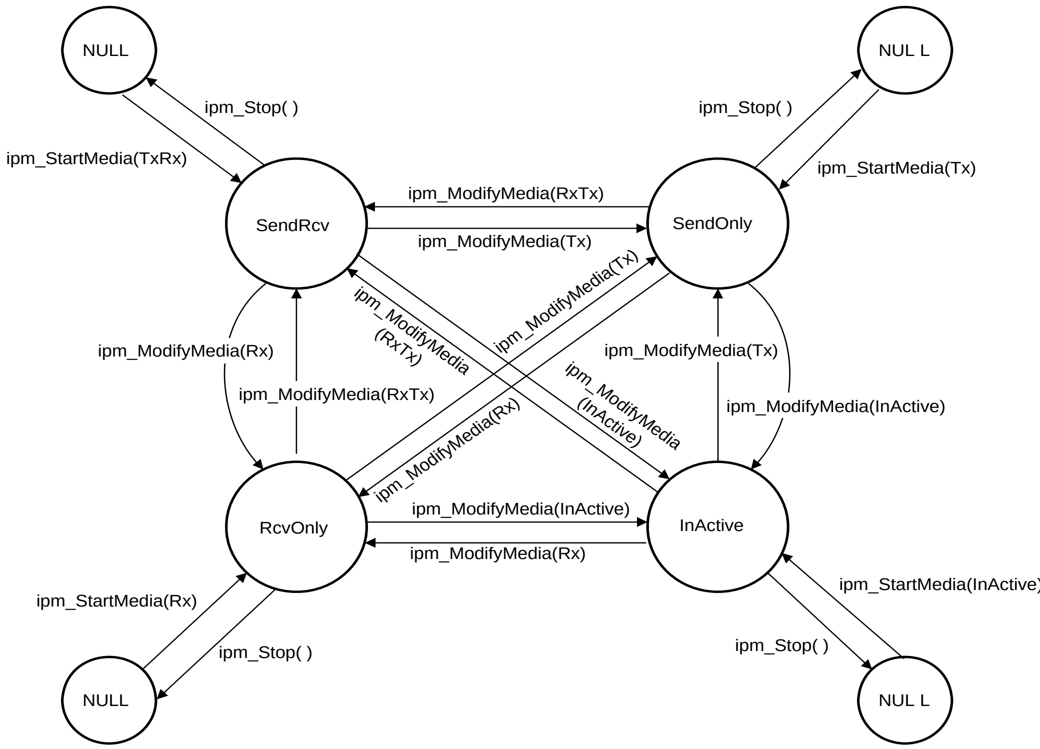
DATA_IP_TDM_BIDIRECTIONAL
full-duplex RTP and RTCP path between IP network and TDM

DATA_IP_INACTIVE
allow RTCP while blocking RTP or T.38 packets

See the *IP Media Library API Library Reference* for detailed information about these functions.

Figure 10 shows the possible end point states for media streaming and how the **ipm_StartMedia( )** and **ipm_ModifyMedia( )** functions are used to transition between those states.

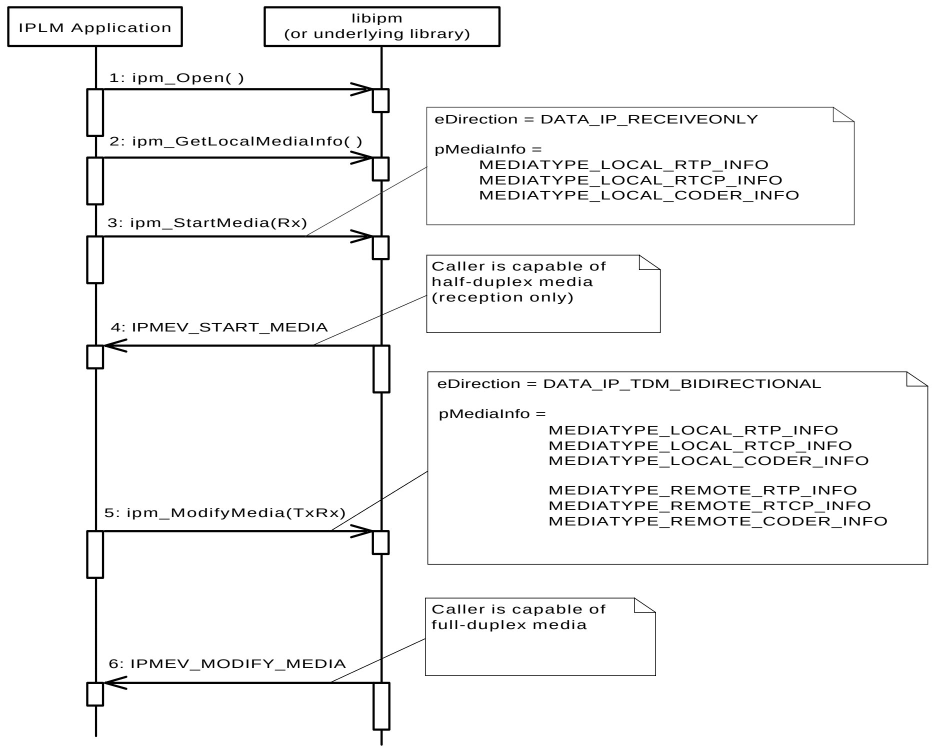
Figure 10. End Point Media Streaming State Transitions



9.3 Application Scenario

Figure 11 shows a scenario in which the **ipm_StartMedia( )** function is used to configure an end point for a half-duplex (receive-only) media stream early in the setup sequence. This media stream can be used to receive call setup information, for example call progress tones. Later, the **ipm_ModifyMedia( )** function can be used to reconfigure the end point for full-duplex media streaming.

Figure 11. Half- and Full-Duplex Media Streaming Application Scenario



9.4 Code Example

The following sample code demonstrates how to reconfigure an end point from full-duplex media streaming to half-duplex (send only) media streaming.

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

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

void main()
{
   /*
   .
   .
   Main Processing
   .
   .
   */

   /*
   Set the media properties for a remote party using IP device handle, nDeviceHandle.
   ASSUMPTION: A valid nDeviceHandle was obtained from prior call to ipm_Open().
   */
```

```
IPM_MEDIA_INFO MediaInfo;
MediaInfo.unCount = 4;

MediaInfo.MediaData[0].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTP_INFO;
MediaInfo.MediaData[0].mediaInfo.PortInfo.unPortId = 2328;
strcpy(MediaInfo.MediaData[0].mediaInfo.PortInfo.cIPAddress, "111.21.0.9");

MediaInfo.MediaData[1].eMediaType = MEDIATYPE_AUDIO_REMOTE_RTCP_INFO;
MediaInfo.MediaData[1].mediaInfo.PortInfo.unPortId = 2329;
strcpy(MediaInfo.MediaData[1].mediaInfo.PortInfo.cIPAddress, "111.21.0.9");

MediaInfo.MediaData[2].eMediaType = MEDIATYPE_AUDIO_REMOTE_CODER_INFO;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eCoderType = CODER_TYPE_G711ULAW64K;
MediaInfo.MediaData[2].mediaInfo.CoderInfo.eFrameSize = (eIPM_CODER_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_AUDIO_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_SYNC) == -1)
{
   printf("ipm_StartMediaInfo failed for device name = %s with error = %d\n",
   ATDV_NAMEP(nDeviceHandle), ATDV_LASTERR(nDeviceHandle));
   /*
   .
   .
   Perform Error Processing
   .
   .*/
}
/*
.
. Continue processing, change destination and direction
.
*/

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

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

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

```
    /*
    .
    .
    continue processing
    .
    */

}
```

Building Applications 10

This chapter contains information on how to compile and link your IPML applications under the Linux* and Windows* operating systems. The information is presented in the following topics:

10.1 Compiling and Linking under Linux

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

10.1.1 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

10.1.2 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
: The 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.

10.2 Compiling and Linking under Windows

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

10.2.1 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

10.2.2 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 `-lgc` argument to the system linker.

libdxxxmt.lib
Required only if the application uses R4 voice library functions directly, for example, **dx_play( )**.

libsrlmt.lib
The Standard Runtime Library (SRL) is mandatory.

Index

J

L

M

N

O

P

Q

R

S

T