9 Building Applications ................................................................. 45
  9.1 Compiling and Linking under Linux .................................................. 45
    9.1.1 Include Files ................................................................. 45
    9.1.2 Required Libraries ......................................................... 45
  9.2 Compiling and Linking under Windows .............................................. 46
    9.2.1 Include Files ................................................................. 46
    9.2.2 Required Libraries ......................................................... 46
Index .................................................................................. 47
Contents

Figures

1 IP Media Architecture ................................................................. 12
2 IP Media Channel State Diagram .................................................. 17
3 In-Band DTMF Mode Scenario Diagram ....................................... 25
4 RFC 2833 Scenario Diagram. ........................................................... 26
5 Out-of-Band DTMF Mode Scenario Diagram .................................. 27
6 T.38 Fax Scenario Diagram ............................................................ 30
7 QoS Scenario 1: Brief Alarm Condition ........................................... 37
8 QoS Scenario 2: True Alarm Condition ............................................ 38
9 QoS Scenario 3: Alarm Condition Cleared ....................................... 39
Tables

1. Quality of Service Parameter Defaults for IPT Series Boards .................. 33
2. Quality of Service Parameter Defaults for DM/IP Series Boards .................. 33
## Revision History

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

<table>
<thead>
<tr>
<th>Document No.</th>
<th>Publication Date</th>
<th>Description of Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-1834-003</td>
<td>September 2003</td>
<td>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</td>
</tr>
<tr>
<td>05-1834-002</td>
<td>November 2002</td>
<td>Application Development Guidelines chapter: Added Using T.38 Fax section</td>
</tr>
<tr>
<td>05-1834-001</td>
<td>September 2002</td>
<td>Initial version of this document</td>
</tr>
</tbody>
</table>
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 series and and Intel NetStructure® IPT series hardware 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.
About This Publication

• 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, “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
• http://developer.intel.com/design/telecom/support/ (for technical support)
• http://www.intel.com/design/network/products/telecom/ (for product information)
Product Description

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

- **Features**
- **Architecture**
- **Introduction to the IP Media Library**
- **Relationship with Global Call Library**
- **Standard Runtime Library Support**
- **Media Channel Device Naming**

### 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.
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
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**

![State Diagram Image]

**NOTE:** The other functions in the IP Media library can be called from any state. They do not cause a state change.
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.
Event Handling

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()`.
Error Handling
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, \texttt{ipm\_SendDigits()} is used when necessary to transmit a DTMF digit towards the TDM bus.

\textit{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, your application must enable the EVT\_RFC2833 event via a call to \texttt{ipm\_EnableEvents()}. Upon receiving the IPMEV\_RFC2833\_SIGNAL\_RECEIVED event, the DTMF information is contained in the IPM\_RFC2833\_SIGNAL\_INFO structure. The application must use \texttt{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 \texttt{ipm\_SetParm()} and the IPM\_PARM\_INFO data structure.

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

- \texttt{DTMFXFERMODE\_INBAND}
  - DTMF digits are sent and received in-band via standard RTP transcoding. This is the default mode when a channel is opened.
  - \textit{Note:} In-band mode cannot be used when using low bit-rate (LBR) coders.

- \texttt{DTMFXFERMODE\_RFC2833}
  - DTMF digits are sent and received in the RTP stream as defined in RFC 2833.

- \texttt{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 \texttt{ipm\_SetParm()}:  

- \texttt{PARMCH\_RFC2833\_EVT\_TX\_PLT}
  - Identifies the transmit payload type. The value range for this field is 96 to 127.

- \texttt{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:

   ```c
   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.

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:
DTMF Handling

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

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

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

```c
IPM_PARM_INFO parmInfo;
unsigned long ulParmValue = 101;
parmInfo.eParm = PARMCH_RFC2833EVT_TX_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**

```
Application

chdev = ipm_Open("ipmB1C1",NULL, EV_SYNC)

ipm_SetParm(chdev, PARMCH_DTMFXFERMODE = DTMFXFERMODE_RFC2833)

ipm_SetParm(chdev,PARMCH_RFC2833EVT_TX_PLT = 101)

ipm_SetParm(chdev,PARMCH_RFC2833EVT_RX_PLT = 101)

Set named event payload on both send and receive sides. Only named event is supported.

IPML Interface
```
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:

   ```c
   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**

   ![Diagram showing the process of setting out-of-band mode](image-url)
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 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.
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.

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("ipmBiCl",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

Application

```c
chdev = ipm_Open("ipmB1C1", NULL, EV_SYNC)
```

```c
ipm_EnableEvents(chDev, *pEvents)
```

Enable T.38 Fax notification events,
*pEvents = IPMEV_T38CALLSTATE, IPMEV_FAXTONE

```c
ipm_GetLocalMediaInfo(chDev,&MediaInfo)
```

Get the local RTP information,
MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_RTP_INFO

```c
ipm_StartMedia(chDev)
```

Start an RTP call when the APP has the remote IP information.

APP receives a FAX tone Event and decides to switch to FAX

```c
ipm_Stop(chDev,STOP_MEDIA)
```

Get the local T.38 information,
MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_UDPTL_T38_INFO

Start a T.38 FAX session when the APP has the remote T.38 IP information.

FAX SESSION STARTED

```c
ipm_StartMedia(chDev)
```

```
SRL
```

Enable T.38 Fax notification events,
*pEvents = IPMEV_T38CALLSTATE, IPMEV_FAXTONE

```c
ipm_GetLocalMediaInfo(chDev,&MediaInfo)
```

Get the local RTP information,
MediaInfo.MediaData[0].eMediaType = MEDIATYPE_LOCAL_RTP_INFO

```
IPML Interface
```

```c
ipm_StartMedia(chDev)
```

Start an RTP call when the APP has the remote IP information.

```
Application
```

APP receives a FAX completion Event, reason code
= T38CALLSTATE_DISCONNECT, and stops the FAX session

```c
ipm_Stop(chDev,STOP_MEDIA)
```

```
SRL
```

```
IPML Interface
```

```
Application
```
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.
Quality of Service (QoS) Alarms

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 Series boards only.

QOSTYPE_ROUNDTRIPLATENCY
   QoS alarm for excessive RTP packet latency
   Note: This alarm is supported on IPT Series boards only.

Note: Intel NetStructure IPT Series 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 Series 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 Series 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 Series Boards

<table>
<thead>
<tr>
<th>QoS Type</th>
<th>Time Interval (ms)</th>
<th>Debounce On (ms)</th>
<th>Debounce Off (ms)</th>
<th>Fault Threshold 1</th>
<th>% Success Threshold</th>
<th>% Fail Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jitter</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30 (ms)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lost Packets</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100 (%)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Round-trip Latency</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>950 (ms)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 Series Boards

<table>
<thead>
<tr>
<th>QoS Type</th>
<th>Time Interval (ms)</th>
<th>Debounce On (ms)</th>
<th>Debounce Off (ms)</th>
<th>Fault Threshold 1</th>
<th>% Success Threshold</th>
<th>% Fail Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jitter</td>
<td>5000</td>
<td>20000</td>
<td>60000</td>
<td>60 (ms)</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVT_JITTER</td>
<td>event indicating excessive jitter</td>
</tr>
</tbody>
</table>


Quality of Service (QoS) Alarms

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

EVT_ROUNDTRIPTLATENCY
   event indicating excessive RTP packet latency (supported on Intel NetStructure IPT Series 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 \texttt{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 \texttt{IPM\_QOS\_THRESHOLD\_INFO} structure with desired values. This structure contains one or more \texttt{IPM\_QOS\_THRESHOLD\_DATA} structures. Note that you must explicitly specify the value for every parameter in the \texttt{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 \texttt{ipm\_SetQoSThreshold()} to use the QoS parameter values set in step 1b.

2. Enable QoS alarms and start media streaming:
   a. Call \texttt{ipm\_EnableEvents()} to enable QoS monitoring for a list of alarm types.
   b. Call \texttt{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 \texttt{sr\_getevtttype()} to return the event type.
   b. Use Standard Runtime Library API functions such as \texttt{sr\_getevtdatap()} to query the \texttt{IPM\_QOS\_ALARM\_DATA} structure to learn whether the alarm state is on or off.

Note: For DM/IP Series boards, the system sends a QoS alarm event containing \texttt{ALARM\_STATE\_ON} when the fault threshold is exceeded and sends a QoS alarm event containing \texttt{ALARM\_STATE\_OFF} when the threshold returns to the
programmed level. For IPT Series 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 Series boards. IPT Series 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
```
Quality of Service (QoS) Alarms

For alarm-off debouncing:
\[
\text{count} = \text{int}(\frac{\text{unDebounceOff}}{\text{unTimeInterval}}) \times \left(\frac{\text{unPercentSuccessThreshold}}{100}\right) \\
= \text{int}(\frac{4000}{1000}) \times \left(\frac{50}{100}\right) \\
= \text{int}(4 \times 0.5) \\
= 2
\]

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 (\text{unTimeInterval} parameter), or every 1 second in this case. When the jitter exceeds the 60ms fault threshold (\text{unFaultThreshold} parameter), the debounce on timer is kicked off (\text{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 (\text{unDebounceOn} parameter), the debounce on window. If the jitter is below the 60ms fault threshold for more than 50 percent of the time (\text{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.
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\(^{(\text{unTimeInterval parameter})}\), or every 1 second in this case. When the jitter exceeds the 60ms fault threshold\(^{(\text{unFaultThreshold parameter})}\), the debounce on timer is kicked off\(^{(\text{unDebounceOn parameter})}\). 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\(^{(\text{unDebounceOn parameter})}\). If the jitter exceeds the 60ms fault threshold for more than 50 percent of the time\(^{(\text{unPercentFailThreshold parameter})}\) 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.
Quality of Service (QoS) Alarms

Figure 8. QoS Scenario 2: True Alarm Condition

QoS parameters:
- time interval = 1 sec
- debounce on = 4 sec
- debounce off = 4 sec
- fault threshold = 60 ms
- % success threshold = 50 %
- % fail threshold = 50 %

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 the 13th 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.
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 and lost packets (which is only supported on IPT Series 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.

```c
#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};

    // Your code here...
}
```

Figure 9. QoS Scenario 3: Alarm Condition Cleared

QoS parameters:
- time interval = 1 sec
- debounce on = 4 sec
- debounce off = 4 sec
- fault threshold = 60 ms
- % success threshold = 50 %
- % fail threshold = 50 %
Quality of Service (QoS) Alarms

// 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.
//    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
     .
     */
}

*/
Quality of Service (QoS) Alarms

/*
  . Continue Processing
  */

// Application 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", (l_pAlarm->eAlarmState? "On": "Off"));
        break;
      }
8.8 Network Failure Alarm (IPT Series Boards Only)

Intel NetStructure® IPT Series 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 Series 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.
Building Applications

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:

- Compiling and Linking under Linux ......................................................... 45
- Compiling and Linking under Windows ..................................................... 46

9.1 Compiling and Linking under Linux

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

9.1.1 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

9.1.2 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.

## 9.2 Compiling and Linking under Windows

The following topics discuss compiling and linking requirements:

- Include Files
- Required Libraries

### 9.2.1 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

### 9.2.2 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.

**libdxmxmt.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

A
alarm state 34
application development guidelines 23
architecture 11
ATDV_ERRMSGP() 19, 21
ATDV_LASTERR() 19, 21
ATDV_NAMEP() 20
ATDV_SUBDEVS() 20

B
building applications 45

C
compiling and linking
under Linux 45
under Windows 46

D
DTMF
detection 28
generation 28
DTMF mode 24
in-band 25
out-of-band 27
RFC 2833 25
DTMF Modes 24
dx_play() 45, 46

E
required libraries 46
error handling 21
event handling 19
events
QoS 33

F
features 11

G
gc_Listen() 23
gc_OpenEx() 45, 46
generating DTMF 28
Global Call library
relationship with IP media 12
Global Call library, relation with 12

I
in-band mode 25
scenario diagram 25
include files 45, 46
IP media
architecture 12
relationship to Global Call 12
standard runtime library support 13
IP media software
media device naming 13
ipm_Close() 13
IPM_DIGIT_INFO 28
ipm_DisableEvents() 35
ipm_EnableEvents() 24, 28, 34
ipm_GetQoSThreshold() 34
ipm_Listen() 23
ipm_Open() 13, 25, 27
IPM_PARM_INFO 24
IPM_QOS_THRESHOLD_DATA 32
IPM_QOS_THRESHOLD_INFO 34
ipm_ReceiveDigits() 23, 27
IPM_RFC2833_SIGNALID_INFO 28
ipm_SendDigits() 24, 28
ipm_SendRFC2833SignalIDToIP() 28
ipm_SetParm() 24, 25, 27
ipm_SetQoSThreshold() 34
ipm_StartMedia() 34
ipm_Stop() 35
IPMEV_QOS_ALARM_DATA 34

J
jitter 32
lost packets 32

media channel device naming 13
mode
  DTMF 24

notification of DTMF detection 28

out-of-band mode 27
  scenario diagram 27

programming models 15

QoS
  alarm and alarm recovery scenarios 35
  alarm state 34
  alarm types 31
  description 31
  events 33
  example code 39
  parameters 32, 36
  threshold attributes 32, 36
  using alarms 34
  quality of service (QoS) 31

required libraries 45
RFC 2833 mode 25
RFC 2833 scenario diagram 26
round-trip latency 32

setting DTMF parameters 24
setting in-band mode 25
setting out-of-band mode 27
setting RFC 2833 mode 25
sr_dishdlr() 19
sr_enbhdlr() 19
sr_getboardcnt() 13
sr_getevtdev() 19
sr_getevttytype() 19
sr_waitevt() 19
sr_waitevtEx() 19
SRL event management functions 19
SRL standard attribute functions 19
Standard Runtime Library support 13
state models 17

T.38 fax server 29