IP Multicast Server (IPML)

Demo Guide

November 2003

05-1824-002

Publication Date: November 2003

Document Number: 05-1824-002

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

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

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

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

Contents

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
05-1824-002	November 2003	Global changes: Changed file directory path. Removed "_r4" from file names Figure 1: replaced TRACE with FUNC_TRACE and STATE_TRACE
05-1824-001	October 2002	Initial production version of document.

About This Publication

The following topics provide information about this guide:

Purpose

This guide provides information on the IP Multicast Server (IPML) demo that is available with your Intel® Dialogic® system release. This guide describes the demo, its requirements, and details on how it works.

Intended Audience

This guide is intended for application developers who will be developing an IP multicast server application using the IPML API. Developers should be familiar with the C++ programming language and the Windows* programming environments.

This information is intended for:

- Distributors
- 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.

This publication assumes that you are familiar with the Windows operating system and the C++ programming language.

The information in this guide is organized as follows:

- Chapter 1, “Demo Description” introduces you to the demo and its features

- Chapter 2, “System Requirements” outlines the hardware and software required to run the demo
- Chapter 3, “Preparing to Run the Demo” describes the preparations required before running the demo
- Chapter 4, “Running the Demo” describes how to run the demo
- Chapter 5, “Demo Details” provides details on how the demo works
- Chapter 6, “Demo State Machines” describes the demo state machines

Related Information

See the following for more information:

- The online Release Update for your specific system release for information on problems fixed, known problems and workarounds, and documentation updates.
- *Intel DM3 Architecture PCI Products on Windows Configuration Guide*
- *Global Call IP Technology Guide*
- *http://developer.intel.com/design/telecom/support/* for technical support
- *http://www.intel.com/design/network/products/telecom/* for product information

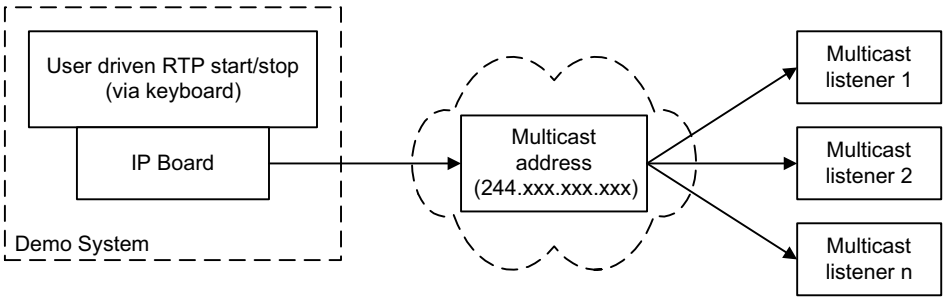
Demo Description 1

This chapter provides a brief description of the IP Multicast Server (IPML) demonstration program.

The IP Multicast Server (IPML) demo is an object-oriented host-based application that illustrates how to build a simple Internet multicast server application using the IPML API. It allows an IP server to use RTP multicasting to continuously deliver an RTP stream to a multicast IP address. At the edge of the IP network, IP gateways are used to listen to the RTP streams. Multicasting allows several callers to access the same information, such as weather forecasts, stock market information, etc. Any client, such as the IP Multicast Client (IPML) demo supplied with this release or a client such as NetMeeting*, can be used together with this demo to create a complete multicast server/client application. The client must be able to access standard multicast IP addresses (224.0.0.0 to 239.255.255.255) and play a *.vox* file.

The basic demo system is shown in Figure 1:

Figure 1. IP Multicast Server (IPML) Demo System



The IP Multicast Server (IPML) demo supports the following features:

- play a vox file to a multicast address
- define system environment via a configuration file
- specify run-time options via command line switches
- print output log files to a file
- print selected log files to the monitor
- change debug levels via keyboard during run-time

The IP Multicast Server (IPML) demo does not have a call control component, and therefore does not support any call control related features.

The IP Multicast Server (IPML) demo is a cross-OS demo, running under the Windows or Linux environments. Most of the differences in the environments are handled directly by the programming interface and are transparent to the user. Other differences, due to inherent

differences in the operating systems, are handled by the Platform Dependency Library (PDL). For more information about the PDL refer to the source code in the *pdl_win* or *pdl_linux* directories.

System Requirements 2

This chapter discusses the system requirements for running the IP Multicast Server (IPML) demo. It contains the following topics:.

2.1 Hardware Requirements

To run the IP Multicast Server (IPML) demo, you need:

- Intel® NetStructure™ DM/IP Series board
- IP network cable

For other hardware requirements, such as memory requirements, see the Release Guide for the system release you are using.

2.2 Software Requirements

To run the IP Multicast Server (IPML) demo, you need the Intel® Dialogic® System Release 6.0 for Windows software. For a list of operating system requirements see the Release Guide for the system release you are using.

See Section 3.2, “Compiling and Linking”, on page 17 for a list of compilers that may be used with this demo. Using a non-supported compiler may cause unforeseen problems in running the demo.

Preparing to Run the Demo 3

This chapter discusses the preparations necessary to run the IP Multicast Server (IPML) demo. It provides information about the following topics:

3.1 Editing Configuration Files

Before running the IP Multicast Server (IPML) demo, modify the *multicastserver.cfg* file to reflect your system environment. Use a text editor and open the file from:

- *C:\Program Files\dialogic\demos\ipdemo\multicastserver\release*

Editing the multicastserver.cfg Configuration File

Below is an example of the *multicastserver.cfg* file. Update the following information:

DestinationIP
: The multicast address the audio file is transmitted to (224.0.0.0 to 239.255.255.255). The same destination IP address may be used for multiple channels if different destination RTP ports are used.

DestinationRTP
: The RTP port the audio file is transmitted to (starts with 2326, even numbers only). The same RTP port may be used for multiple channels if different destination IP address are used.

TransmitFile
: The filename of the audio file to be transmitted. The IP Multicast Server (IPML) demo currently supports *.vox* files only.

TxCoderType
: The type of coder used to broadcast. See the *Global Call IP Technology Guide* for specific information about coder support in this release. The IP Multicast Server (IPML) demo recognizes the following coder name spellings:
  - g711Alaw
  - g711Mulaw
  - gsm
  - gsmEFR
  - g7231_5_3k
  - g7231_6_3k
  - g729a
  - g729ab

TxCoderFramesPerPkt
: Specify the number of frames per packet for the selected coder. See the *Global Call IP Technology Guide* for specific information about coder support in this release.

TxCoderFrameSize
: Specify the frame size for the selected coder. See the *Global Call IP Technology Guide* for specific information about coder support in this release.

TxCoderVAD
: Specify if VAD is active. See the *Global Call IP Technology Guide* for specific information about coder support in this release.

TxPayload
: Describes the static payload type values for the PT field of the RTP data header as described in RFC 1890. See the *Global Call IP Technology Guide* for specific information about coder support in this release.

TxRedPayload
: Describes the static payload type value for the first redundant frame within the packet. This parameter must be set in order for the system to identify the redundant packets. See the *Global Call IP Technology Guide* for specific information about coder support in this release.

The following example shows the configuration file. Due to the length of the file, the example shows three channels only.

```
# Each channel represents a specific service.
# The data that should be configured includes:
# The destinationIP - The multicast address for transmission (224.0.0.0 - 239.255.255.255).
# The DestinationRTP - The RTP port for transmission (start with 2326, just the even number).
# TransmitFile - The file that will be transmitted.
# Tx Coder values.
# The multicast address + RTP port should be DIFFERENT for each channel.
# e.g: if the multicast address is the same in channel 1 and channel 2
# then the RTP port should be different.

Channel = 1
{
   DestinationIP = 224.0.0.1
   DestinationRTP = 2370
   TransmitFile = MC_file1.vox
   TxCoderType = g711mulaw
   TxCoderFramesPerPkt = 1
   TxCoderFrameSize = 30
   TxCoderVAD = 0
   TxPayload = 0
   TxRedPayload = 0
}

Channel = 2
{
   DestinationIP = 224.0.0.2
   DestinationRTP = 2370
   TransmitFile = MC_file2.vox
   TxCoderType = g711mulaw
   TxCoderFramesPerPkt = 1
   TxCoderFrameSize = 30
   TxCoderVAD = 0
   TxPayload = 0
   TxRedPayload = 0
}
```

```
Channel = 3
{
   DestinationIP = 224.0.0.3
   DestinationRTP = 2370
   TransmitFile = MC_file3.vox
   TxCoderType = g711mulaw
   TxCoderFramesPerPkt = 1
   TxCoderFrameSize = 30
   TxCoderVAD = 0
   TxPayload = 0
   TxRedPayload = 0
}
```

3.2 Compiling and Linking

Compile the project within the following environments:

- Visual C++ environment, version 6

To compile the project, put the files in the Dialogic directory under *dialogic\demos\ipdemo\multicastserver.*

Set **multicastserver** as the active project and build in debug mode.

3.3 Selecting PCD/FCD Files

Note: This section refers to Intel® NetStructure™ DM/IP series boards only.

Choose a PCD and matching FCD file that begins with the **ipvs_evr** prefix. Refer to the *Intel DM3 Architecture PCI Products on Windows Configuration Guide* for complete configuration information.

Running the Demo 4

This chapter discusses how to run the IP Multicast Server (IPML) demo. It contains the following topics:

4.1 Starting the Demo

Select **Run** from the Start Menu. The demo executable file can be found in: *C:\Program Files\Dialogic\demos\ipdemo\multicastserver\release\multicastserver.exe*. Click **OK** to run the IP Multicast Server (IPML) demo using the default settings.

4.2 Demo Options

To specify certain options at run-time, launch the demo from a command line, using any of the switches listed in Table 1, "Command Line Switches", on page 19.

Table 1. Command Line Switches

Switch	Action	Default
-c <filename>	Configuration file name	-c multicastserver_r4.exe
-d<*n*>	Sets Debug Level (0-4): • 0-FATAL – used when one or more channels are deadlocked. • 1-ERROR – used when the application receives a failure which doesn't cause the channel to be deadlocked. • 2-WARNING – used when some problem or failure occurred without affecting the channel's usual action. • 3-STATE_TRACE – used to monitor state transitions for Devices • 4-INFO – prints data related to a specific action. • 5-FUNC_TRACE – used at the start of the application entrance or the start of any function. **Note:** Debug level is inclusive; higher levels include all lower levels	-d0 (Fatal)
-e/E	The encoding format in which the .vox files were recorded: • m = mu-law • a = A-law	-em (mu-law)
-h/?	Prints the command syntax to the screen	Off

Table 1. Command Line Switches (Continued)

Switch	Action	Default
-l<*n,…*>	Printouts will be printed into channel log files. If 'all' follows the –l, log files will be created for all available channels. If a list of channels in the following format: C1-C2, C3-C4, C5 follows the –l, log files are created for the channel ranges or specific channels specified in the list. If the "–l" option is not used, prints go to the stdout, for the first 2 channels only (to keep from overloading the CPU, and more convenient for viewing printouts).	Disabled
-m<*n,…*>	Enables printing channel specific information to the monitor, in addition to printing the log file. A maximum of 2 channels may be printed.	Disabled
-n<*n*>	Sets the number of server channels	The lesser of Voice Devices or IP devices

4.3 Using the Demo

This section discusses how to use the demo. It contains the following topics:

- Establishing and Terminating a Call
- Keyboard Commands

4.3.1 Establishing and Terminating a Call

The demo waits for input from the keyboard. Press “e” or “E” to establish a call on each of the configured IP channels. The demo starts the Player and plays the audio file specified in the configuration file in an endless loop for each channel. Each channel plays the specified audio file toward the multicast address defined in the configuration file.

Press “t” or “T” to terminate the call on all the channels. A new call can be established by pressing “e” or “E” again.

Notes: ***1.*** The voice files start with 3 seconds of silence to enable load testing.

2. The demo does not allow terminating a call on a single channel. All channels are terminated by pressing “t” or “T.”

4.3.2 Keyboard Commands

The demo always waits for input from the keyboard. While the demo is running, you may enter any of the following commands:

Table 2. Runtime Keyboard Commands

Command	Function
d<n> or D<n>	Change debug level during runtime
e or E	Establish a call
m or M	Print log files for up to 2 channels to the screen
q or Q or CTRL+C	Terminate the application
t or T	Terminate a call

4.4 Stopping the Demo

The IP Multicast Server (IPML) demo runs until it is terminated. To terminate the demo press “t” or “T” to close all the channels or press “q” or “Q” or “Ctrl+c” to terminate the demo application.

Demo Details 5

This chapter discusses the IP Multicast Server (IPML) demo in more detail. It contains the following topics:

5.1 Files Used by the Demo

This section lists the files used by the demo. It contains the following information:

- Demo Source Code Files
- Utility Files
- PDL Files

5.1.1 Demo Source Code Files

The source code files listed in Table 3 are located in:

- For Windows: *C:\Program Files\dialogic\demos\ipdemo\multicastserver*

Table 3. Source Files Used by the IP Multicast Server (IPML) Demo

Directory	File Name	Purpose
multicastserver	call.cpp	Implements the operations of the Call class
multicastserver	call.h	Function prototype for call.cpp
multicastserver	configuration.cpp	Implements the operations of the Configuration class
multicastserver	configuration.h	Function prototype for configuration.cpp
multicastserver	incfile.h	Function prototype for Global Call and R4 functions
multicastserver	ipmediaboard.cpp	Implements the operations of the IPMediaBoard class
multicastserver	ipmeadiaboard.h	Function prototype for ipmediaboard.cpp
multicastserver	ipmediadevice.cpp	Implements the operations of the IPMediaDevice class
multicastserver	ipmediadevice.h	Function prototype for ipmediadevice.cpp
multicastserver	main.cpp	Contains the main function and the Wait for Key
multicastserver	main.h	Function prototype for main.cpp

Table 3. Source Files Used by the IP Multicast Server (IPML) Demo (Continued)

Directory	File Name	Purpose
multicastserver	multicastserver.ver	Demo version information
multicastserver	r4device.cpp	Implements the operations of the R4Device class
multicastserver	r4debice.h	Function prototype for r4device.cpp
multicastserver	r4logicalboard.cpp	Implements the operations of the R4LogicalBoard class
multicastserver	r4logicalboard.h	Function prototype for r4logicalboard.cpp
multicastserver	resourcemanager.cpp	Implements the operations of the ResourceManager class
multicastserver	resourcemanager.h	Function prototype for resourcemanager.cpp
multicastserver	voiceboard.cpp	Implements the operations of the VoiceBoard class
multicastserver	voiceboard.h	Function prototype for voiceboard.cpp
multicastserver	voicedevice.cpp	Implements the operations of the VoiceDevice class
multicastserver	voicedevice.h	Function prototype for voicedevice.cpp
multicastserver	multicastserver.dsp	Visual C++ project file
multicastserver	multicastserver.dsw	Visual C++ project workspace
multicastserver\release (Windows only)	multicastserver.cfg	Demo configuration file
multicastserver\release (Windows only)	multicastserver.exe	Demo executable

5.1.2 Utility Files

The utility files listed in Table 4 are located in:

- For Windows: *C:\Program Files\dialogic\demos\ipdemo\utilcpp*

Table 4. Utility Files Used by the IP Multicast Server (IPML) Demo

Directory	File Name	Purpose
utilcpp	utilcpp.ver	Utility library version information
utilcpp	log.cpp	Debugging functions
utilcpp	log.h	Function prototype for libdbg.c
utilcpp (Windows only)	utilcpp.dsw	Utility library Visual C++ workspace
utilcpp (Windows only)	utilcpp.dsp	Utility library Visual C++ project file
utilcpp\release (Windows only)	utilcpp.lib	Compiled Utility library

5.1.3 PDL Files

The Windows PDL files listed in Table 5 are located in:
C:\Program Files\dialogic\demos\ipdemo\pdl_win.

Table 5. PDL Files Used by the IP Multicast Server (IPML) Demo - Windows OS

Directory	File Name	Purpose
pdl_win	iptransport.cpp	PDL IP transport functions
pdl_win	iptransport.h	Function prototype for iptransport.cpp
pdl_win	pdl.c	Platform dependency functions
pdl_win	pdl.h	Function prototype for pdl.c
pdl_win	pdl.ver	PDL version information
pdl_win	pdl_win.dsp	PDL Visual C project file
pdl_win	pdl_win.dsw	PDL Visual C workspace
pdl_win\release	pdl_win.lib	Compiled PDL library

5.2 Programming Model Classes

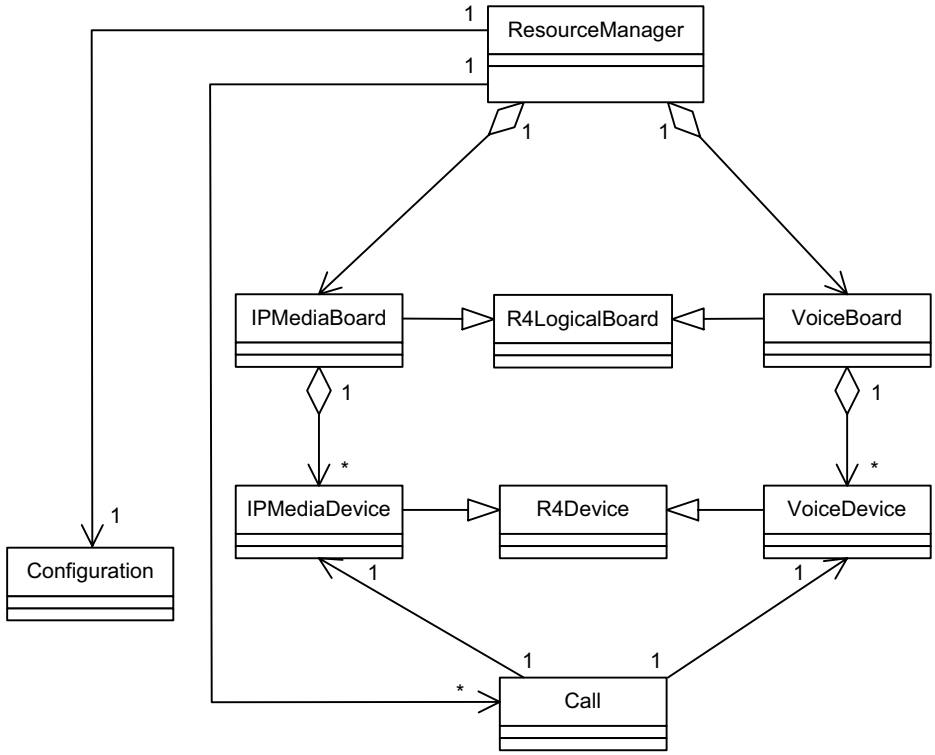
This section presents basic information about the IP Multicast Server (IPML) demo classes. It contains the following information:

- Class Diagram
- Call Class
- Configuration Class
- IPMediaBoard Class
- IPMediaDevice Class
- R4Device Class
- R4LogicalBoard Class
- ResourceManager Class
- VoiceBoard Class
- VoiceDevice Class

5.2.1 Class Diagram

The following class diagram describes the relationship among the classes.

Figure 2. IP Multicast Server (IPML) Class Diagram



5.2.2 Call Class

The main role of the Call class is to control all the resources related to a call. It contains all the resources related to a call and reflects the intersection of the call resource status.

The Call class attributes are described in Table 6. Refer to the source code for method information.

Table 6. Call Class Attributes

Name	Access Privilege	Type	Description
m_pIPMediaDevice	public	IPMediaDevice*	The IPMedia device of the channel
m_pLog	public	Log*	A log instance
m_ptheConfiguration	public	Configuration*	Pointer to the configuration instance of the ResourceManager
m_pVoiceDevice	public	VoiceDevice*	The voice device of the channel
m_channelId	private	unsigned int	The channel identifier
m_currentState	private	E_StateMachine	The current state of the call

5.2.3 Configuration Class

The main role of the Configuration class is to provide an interface to get the needed configuration data. It contains all the needed data structures to parse and save the system configuration (the configuration file and the command line options) and reflects the system configuration to the other classes.

The Configuration class attributes are described in Table 7. Refer to the source code for method information.

Table 7. Configuration Class Attributes

Name	Access Privilege	Type	Description
m_chanInfo	public	ChannelInfo	Array that contains all the channel information from the configuration file, such as Tx coder information, the print to log file flag, and the phone number to call.
m_logFArr	public	char	Array to get the string entered by the user after the -l option
m_logLevel	public	E_LogLevel	The log level
m_userChannels	public	unsigned int	Indicates the number of channels that the demo will work with
m_cfgFile	private	char*	The configuration file name
m_EncodingType	private	int	The encoding type (Mulaw/Alaw)
m_firstSession	private	long	Used to fill the channel information from the configuration file
m_lastSession	private	long	Used to fill the channel information from the configuration file
m_line	private	int	The line currently being parsed in the configuration file
m_logFileFlag	private	int	Flag for using log files, one for each channel (by the -l command line option)
m_stage	private	unsigned char	The stage of parsing the configuration file

5.2.4 IPMediaBoard Class

The main role of the IPMediaBoard class is to manage the IP Media device database. It contains all the IP Media devices available to the system and reflects the IP Media device repository.

The IPMediaBoard class attributes are described in Table 8. Refer to the source code for method information.

Table 8. IPMediaBoard Class Attributes

Name	Access Privilege	Type	Description
m_ipMediaDevices	public	IPMediaDevice	Array of the IPMedia devices found on the IP board

5.2.5 IPMediaDevice Class

The main role of the IPMediaDevice class is to provide IPML functionality for the IP Media device. It represents the IP Media devices and reflects the session state to the other classes.

The IPMediaDevice class attributes are described in Table 9. Refer to the source code for method information.

Table 9. IPMediaDevice Class Attributes

Name	Access Privilege	Type	Description
m_currentState	private	E_StateMachine	The current state of the channel
m_localMediaInfo	private	IPM_MEDIA_INFO	The local media information
m_mediaInfo	private	IPM_MEDIA_INFO	Used in the **startMedia( )** function when calling the function **ipm_SetRemoteMediaInfo( )**. The m_mediaInfo contains: • the local RTP and RTCP information and the local coder information • the remote RTP and RTCP information and the remote coder information
m_mediaStartedFlag	private	bool	Flag to indicate if the media has started
m_remoteMediaInfo	private	IPM_MEDIA_INFO	The remote GW media information

5.2.6 R4Device Class

The main role of the R4Device class is to provide all common functionality for all R4 devices. It is the base class for all R4 line devices that can be opened using **gc_OpenEx( )**. It contains all the common attributes and operations for all R4 devices.

The R4Device class attributes are described in Table 10.

Table 10. R4Device Class Attributes

Name	Access Privilege	Type	Description
m_channelId	protected	unsigned int	The identifier of the channel that the device belongs to
m_handle	protected	unsigned int	The device handle (valid after opening)
m_name	protected	char	The device name, e.g. ipmB1C1
m_pLog	protected	Log*	The device log instance
m_txTimeSlot	protected	unsigned long	The device time slot

5.2.7 R4LogicalBoard Class

The main role of the R4LogicalBoard class is to provide all common functionality for all R4 logical boards. It opens the boards and gets all the information about the devices. The R4LogicalBoardClass is the base class for all R4 logical boards containing the common attributes.

The R4LogicalBoard class attributes are described in Table 11. Refer to the source code for method information.

Table 11. R4LogicalBoard Class Attributes

Name	Access Privilege	Type	Description
m_boardHandle	protected	int	Returned when opening the board by **dx_Open( )** or **dt_Open( )** and used to get the devices found on it by calling the function **ATDV_SUBDEVS( )**, and to close the board.
m_boardName	protected	char	The board name, e.g., ipmB1
m_boardNumber	protected	int	The board number - used in setting the device names found on the board
m_numOfChannelsOnBoard	protected	int	Number of devices available on the board

5.2.8 ResourceManager Class

The main role of the ResourceManager class is to initialize the R4 resources. It manages the system resources and contains the following data:

- all system channels
- configuration object for initialization
- maps R4 device handles to channels

- all detected R4 boards

The ResourceManager class attributes are described in Table 12.

Table 12. ResourceManager Class Attributes

Name	Access Privilege	Type	Description
DeviceHandleToChannel	public	static unsigned int	Maps the devices to channels to enable handling the SRL events. The table is filled in when opening each device.
m_Calls	public	static Call*	Array that contains all the calls used by the application
m_IPBoards	public	IPMediaBoard*	Array that includes the IP boards available in the system
m_maxChannelsToOpen	public	int	The maximum number of channels that the demo will work with (this number is the minimum of devices of each type and the user requested -n option).
m_numOfIPBoards	public	int	The number of the IP boards found in the system
m_numOfVoiceBoards	public	int	The number of the voice boards found in the system
m_pLog	public	static Lot*	A log instance used during initialization. All the printouts are to the monitor - after that it is killed.
m_ptheConfiguration	public	static Configuration*	An instance of the Configuration Class that is used to determine the configuration of the system during initialization
m_VoiceBoards	public	VoiceBoard*	Array that includes the voice boards available in the system

5.2.9 VoiceBoard Class

The main role of the VoiceBoard class is to manage the voice device database. It contains all the voice devices available to the system and reflects the voice device repository.

The VoiceBoard class attributes are described in Table 13. Refer to the source code for method information.

Table 13. VoiceBoard Class Attributes

Name	Access Privilege	Type	Description
m_voiceDevices	public	VoiceDevice	Array of the voice devices found on the voice board

5.2.10 VoiceDevice Class

The main role of the VoiceDevice class is to provide R4 functionality to the voice devices. It represents a real voice resource channel and always reflects its status. It manages all calls related to itself.

The VoiceDevice class attributes are described in Table 14.

Table 14. VoiceDevice Class Attributes

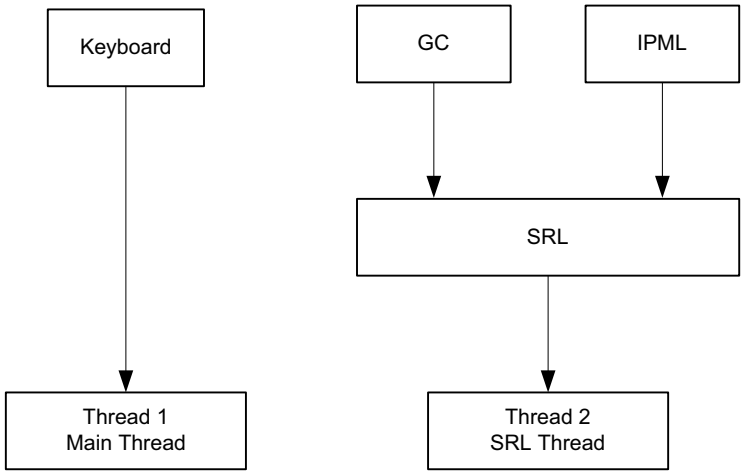
Name	Access Privilege	Type	Description
m_PlayerStopped	public	int	Flag that indicates if the player is stopped
prompt	public	DX_IOTT	Structure that identifies a source or destination for voice data. It is used with **dx_play( )**.

5.3 Threads

The IP Multicast Server (IPML) demo operates with two threads:

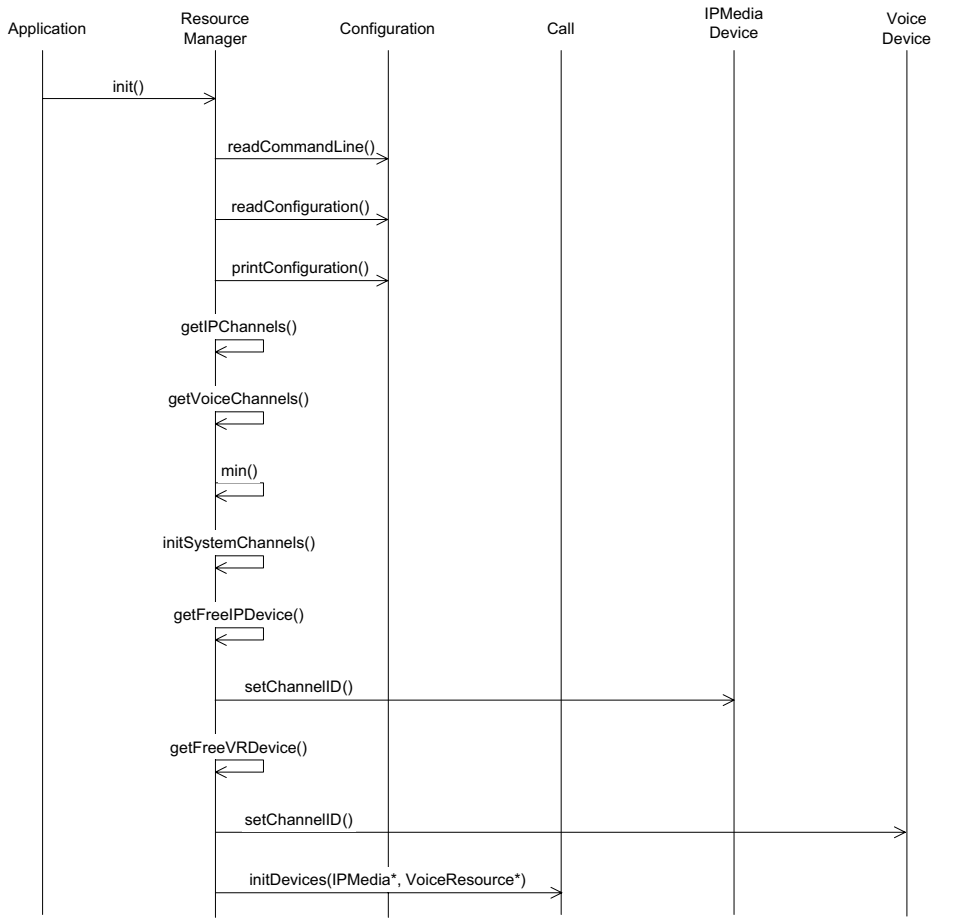
- The first thread (main) is created by the demo application to get the keyboard input
- The second thread is an SRL thread, created as a result of the demo application calling **sr_enblhdlr( )** in Windows. In Linux, the thread must be explicitly created.

Figure 3. Thread Diagram



5.4 Initialization

Figure 4. IP Multicast Server (IPML) System Initialization



This section describes the demo initialization as shown in Figure 4.

1. The application **main( )** function sets up the callback handler, **PDLsr_enbhdlr( )**. The callback handler handles events that it receives from the SRL library. For more details see Section 5.5.3, “Handling SRL Events”, on page 33.
2. The application **main( )** function then calls **resourceManager.init( )**, which does the following:
   a. Reads the command line options
   b. Reads and parse the configuration file and prints the configuration
   c. Gets the resources available in the system:
      – Gets the number of IP channels in the system
      – Gets the number of Voice channels in the system
      – Finds the minimum between the system channels and the user request

d. Looks for a free ipmedia device and returns a pointer to it
e. Opens the ipmedia device and if the open succeeds returns a pointer to it
f. Looks for a free voice device and returns a pointer to it
g. Opens the voice device and if the open success returns a pointer to it
h. Initializes the devices on the channel

3. The application **main( )** function calls **waitForKey( )**, to receive keyboard input

5.5 Event Handling

This section contains the following topics:

5.5.1 Event Mechanism

The IP Multicast Server (IPML) demo uses the SRL mechanism to retrieve events. When an event occurs, SRL calls event handlers automatically. All events are received by the SRL and then passed to the **callback_hdlr( )** function for handling.

In the initialization phase of the demo the **init( )** function sets up the call-back handler, by calling **PDLsr_enbhdlr( )**.

5.5.2 Handling Keyboard Input Events

There is an endless loop **{while(1)}** in the **main( )** function in the *main.cpp* file. In that loop, the application waits forever for a keyboard event by calling the **waitForKey( )** function. The event must be handled immediately and event-specific information should be retrieved before the next call to **waitForKey( )**.

When the next event occurs or when a time-out is reached, the **waitForKey( )** returns and the call-back handler function is called automatically.

5.5.3 Handling SRL Events

When the R4/Global Call event is received, the application performs the following:

1. Get the event device handle, by calling **PDLsr_getevtdev( )**
2. Get the channel number related to the event, from the global array (HandleToChannel[ ])
3. Update the METAEVENT structure by calling **gc_GetMetaEvent( )**
4. Get the event type, by calling **PDLsr_getevttype( )**

Demo State Machines 6

This chapter discusses the IP Multicast Server (IPML) state machines. It contains the following topics:

6.1 Call State Machine

This section describes the Call class state machine. It contains the following topics:

- Call State Machine Description
- Call::callNull State
- Call::callStarted State
- Call::callProceeding State
- Call::callStopped State

6.1.1 Call State Machine Description

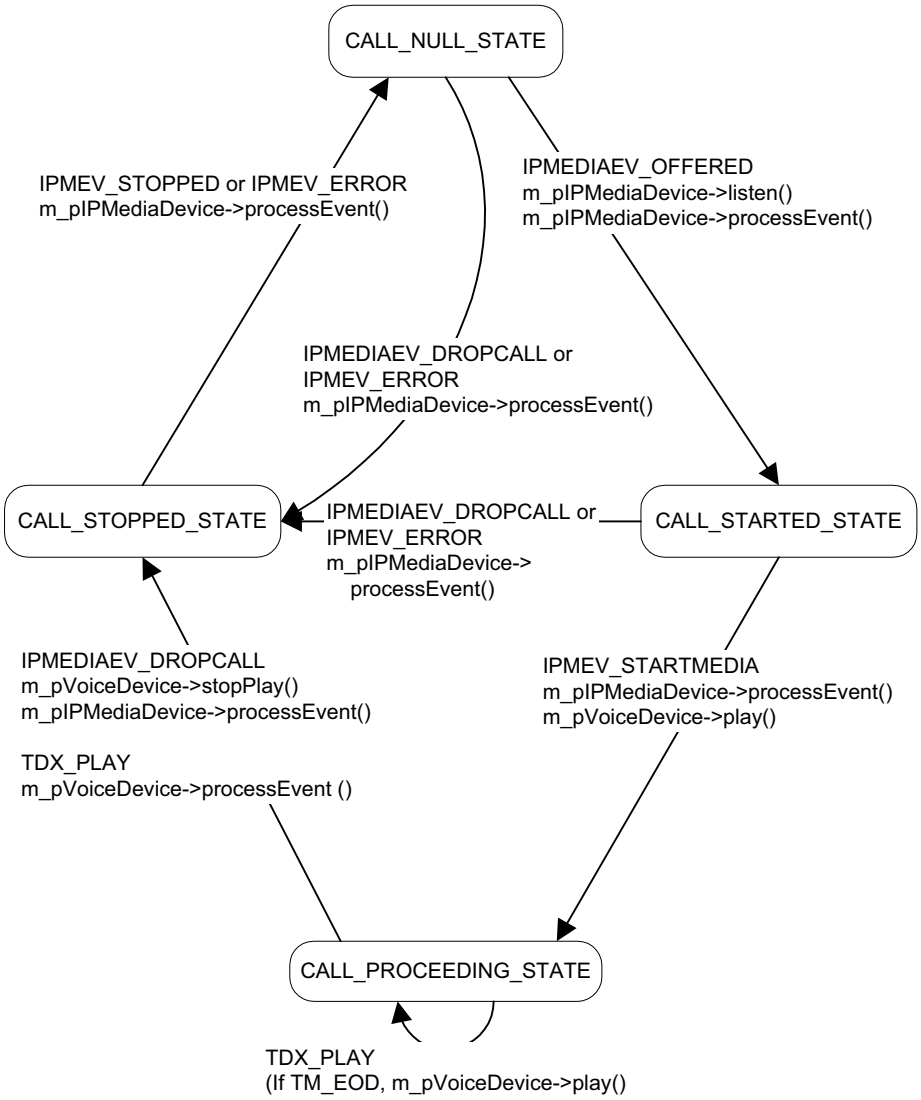
All channels are initialized to the NULL state upon application start.

As soon as an event is received, the event type, the channel number, and the reason for the event (if there is one), are analyzed and the appropriate state machine function is called.

After all the operations are performed within the channel's event state, the state machine function is updated.

The following state diagram describes the call states for the call class.

Figure 5. Call State Machine



6.1.2 Call::callNull State

The application waits for a IPMEDIAEV_OFFERED event in the CALL_NULL_STATE. Upon receiving this event, the application calls **listen( )** from the IPMediaDevice to listen to the media stream. It then calls **processEvent( )** from the IPMediaDevice to process the incoming event. The call state transitions to callStarted.

If, for any reason, the function should fail, the application receives an IPMEDIAEV_DROPCALL or IPMEV_ERROR event. The application calls **processEvent( )** from the IPMediaDevice and the call state transitions to callStopped.

6.1.3 Call::callStarted State

The application waits for an IPMEV_STARTMEDIA event, following the IPMediaDevice call to **startMedia( )** (see Section 6.2, "IPMediaDevice State Machine", on page 37 for more information about the IPMediaDevice state machine). Upon receiving this event, the application calls **processEvent( )** from the IPMediaDevice and then calls **play( )** from the VoiceDevice to begin broadcasting. The call state transitions to callProceeding.

If, for any reason, the function should fail, the application receives a IPMEDIAEV_DROPCALL or IPMEV_ERROR event. The application calls **processEvent( )** from the IPMediaDevice and the call state transitions to callStopped.

6.1.4 Call::callProceeding State

The application waits for an IPMEDIAEV_DROPCALL event. Upon receiving this event, it calls **stopPlay( )** from the VoiceDevice and **processEvent( )** from the IPMediaDevice. The call state transitions to callStopped.

If the application receives a TDX_PLAY event, it determines if the Player was stopped because it reached the end of data or it the call has completed. In the case of end of data, the Player replays the file. In the case of call completion, the application calls **processEvent( )** from the VoiceDevice.

6.1.5 Call::callStopped State

The application waits for an IPMEV_STOPPED or IPMEV_ERROR event. In the case of IPMEV_STOPPED, the application calls **processEvent( )** from the IPMediaDevice and the call state transitions to callNull. In the case of IPMEV_ERROR, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice and the call state transitions to callNull.

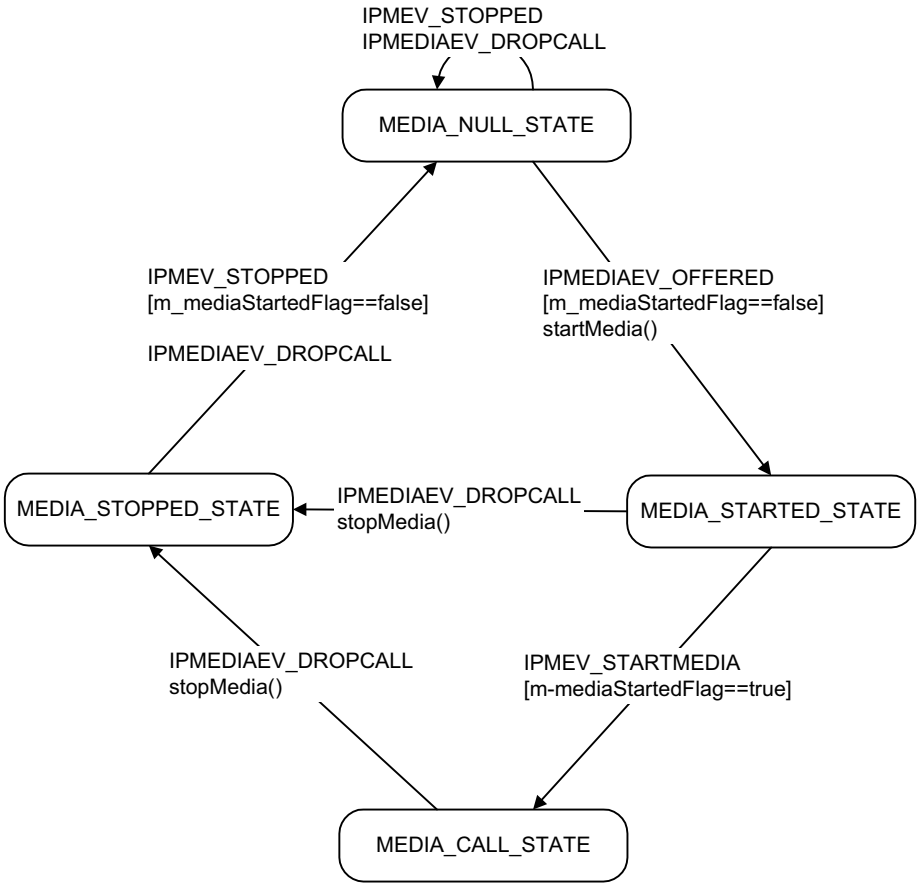
6.2 IPMediaDevice State Machine

This section describes the IPMediaDevice state machine. It contains the following topics:

6.2.1 IPMediaDevice State Machine Description

The following state diagram describes the states for the IPMediaDevice class.

Figure 6. IPMediaDevice State Machine



6.2.2 IPMediaDevice::mediaNull State

The application waits for a IPMEDIAEV_OFFERED event in the mediaNull state. Upon receiving this event it calls **startMedia( )** from the IPMediaDevice. The state transitions to mediaStarted.

If the application receives IPMEV_STOPPED or IPMEDEV_DROPCALL, it ignores these events and continues to wait for IPMEDIAEV_OFFERED.

6.2.3 IPMediaDevice::mediaStarted State

The application waits for an IPMEV_STARTMEDIA event. Upon receipt of this event, it sets the **m_mediaStartedFlag** to true and the state transitions to the mediaCall state.

If the application receives IPMEDIAEV_DROPCALL, it calls **stopMedia( )** from the IPMediaDevice and the state transitions to mediaStopped.

6.2.4 IPMediaDevice::mediaCall State

The application waits for an IPMEDIAEV_DROPCALL event. Upon receipt of this event, it calls **stopMedia( )** from the IPMediaDevice and the state transitions to mediaStopped.

6.2.5 IPMediaDevice::mediaStopped State

The application waits for an IPMEV_STOPPED event. Upon receipt of this event, it sets the **m_mediaStartedFlag** to false and the state transitions to mediaNull.

If the application receives an IPMEDIAEV_DROPCALL event, the state transitions to mediaNull.

Glossary

Codec: see COder/DECoder

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

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

DTMF: Dual-Tone Multi-Frequency

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

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

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

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

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

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

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

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

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

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

IFP: Internet Facsimile Protocol

IFT: Internet Facsimile Transfer

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

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

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

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

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

ITU: See International Telecommunications Union.

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

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

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

PSTN: see Public Switched Telephone Network

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

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

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

RTCP: Real Time Control Protocol

RTP: Real Time Protocol

SIP: Session Initiation Protocol: an Internet standard specified by the Internet Engineering Task Force (IETF) in RFC 2543. SIP is used to initiate, manage, and terminate interactive sessions between one or more users on the Internet.

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

TCP: see Transmission Control Protocol

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

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

UDP: see User Datagram Protocol

UDPTL: Facsimile UDP Transport Layer protocol

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

VAD: Voice Activity Detection

Index