IP Multicast Client (IPML)

Demo Guide

November 2003

05-1839-001

Publication Date: November 2003

Document Number: 05-1839-001

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
05-1839-001	November 2003	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 Client (IPML) demo that is available with your Intel® Dialogic® system release. This guide describes the demo, its requirements, and provides details on how it works.

Intended Audience

This guide is intended for application developers who will be developing an IP multicast client application using the IPML API. Developers should be familiar with the C++ programming language and either the Windows or Linux 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 or Linux 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 up-to-date information on fixed problems, 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 Client (IPML) demonstration program.

The IP Multicast Client (IPML) demo is an object-oriented host-based application that illustrates how to build a simple Internet multicast client application using the IPML API. It allows an IP gateway to listen to RTP streams broadcast from a multicast IP address. Multicasting allows several callers to access the same information, such as weather forecasts, stock market information, etc. The demo source code can be used as sample code for those who want to begin developing an application from a working application.

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

- configuration file
- command line options
- output log files
- printing to the monitor
- Digital PSTN

The following features are not supported by this demo application:

- Analog PSTN
- UII message
- NonStdCmd message
- NonStdParm data
- Q.931Facility message

The IP Multicast Client (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 Client (IPML) demo. It contains the following topics:.

2.1 Hardware Requirements

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

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 Client (IPML) demo. It provides information about the following topics:

3.1 Editing Configuration Files

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

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

Editing the multicastclient_r4.cfg Configuration File

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

channel_of_server
: Represents a server channel and should be configured to match the server data.

MulticastListenAddress
: The multicast address the file is transmitted from (224.0.0.0 to 239.255.255.255). It should match the server DestinationIP value.

DestinationRTP
: The RTP port the file is transmitted from (starts with 2326, even numbers only). It should match the server DestinationRTP data.

RxCoderType
: The type of coder used to receive. The Rx coder values should match the Tx coder values of the specific multicast server. See the online Release Update for your software release for specific information about coder support in this release. The IP Multicast Client (IPML) demo recognizes the following coder name spellings:

  - g711Alaw
  - g711Mulaw
  - gsm
  - gsmEFR
  - g7231_5_3k
  - g7231_6_3k
  - g729a
  - g729ab

RxCoderFramesPerPkt
: Specify the number of frames per packet for the selected coder. See the online Release Update for your software release for specific information about coder support in this release.

RxCoderFrameSize
: Specify the frame size for the selected coder. See the online Release Update for your software release for specific information about coder support in this release.

RxCoderVAD
: Specify if VAD is active. See the online Release Update for your software release for specific information about coder support in this release.

RxPayload
: Describes the static payload type values for the PT field of the RTP data header as described in RFC 1890. See the online Release Update for your software release for specific information about coder support in this release.

RxRedPayload
: 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 online Release Update for your software release 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.

```
# In order to listen to a server by using a multicast address:
# Each "channel_of_server" represents a server channel and should be configured
# to match the server data.
# e.g:
# The MulticastListenAddress value should match the server DestinationIP value.
# The DestinationRTP should match the server's DestinationRTP data.
# The Rx coder values should match the Tx coder values of the specific server.

Channel = 1 - 30
{
   pstnProtocol = T1 isdn
}

channel_of_server = 1
{
   MulticastListenAddress = 224.0.0.1
   DestinationRTP = 2370
   RxCoderType = g711alaw
   RxCoderFramesPerPkt = 1
   RxCoderFrameSize = 30
   RxCoderVAD = 0
   RxPayload = 0
   RxRedPayload = 0
}

channel_of_server = 2
{
   MulticastListenAddress = 224.0.0.2
   DestinationRTP = 2372
   RxCoderType = g711alaw
   RxCoderFramesPerPkt = 1
```

```
   RxCoderFrameSize = 30
   RxCoderVAD = 0
   RxPayload = 0
   RxRedPayload = 0
}

channel_of_server = 3
{
   MulticastListenAddress = 224.0.0.3
   DestinationRTP = 2374
   RxCoderType = g711alaw
   RxCoderFramesPerPkt = 1
   RxCoderFrameSize = 30
   RxCoderVAD = 0
   RxPayload = 0
   RxRedPayload = 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\multicastclient*.

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

3.3 Selecting PCD/FCD Files

Choose a PCD and matching FCD file that begins with the **ipvs_evr** prefix.

Running the Demo 4

This chapter discusses how to run the IP Multicast Client (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\multicastclient\release\multicastclient.exe*. Click **OK** to run the IP Multicast Client (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 multicastclient_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)
-h or ?	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 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

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
c or C	Print channel statistics
d<n> or D<n>	Change debug level during runtime
m or M	Print log files for up to 2 channels to the screen
q or Q	Terminate the application

4.4 Stopping the Demo

The IP Multicast Client (IPML) demo runs until it is terminated. To terminate the demo press "q" or "Q" to terminate the demo application.

Demo Details 5

This chapter discusses the IP Multicast Client (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:

- *C:\Program Files\dialogic\demos\ipdemo\multicastclient*

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

Directory	File Name	Purpose
multicastclient	channel.cpp	Implements the operations of the Channel class
multicastclient	channel.h	Function prototype for channel.cpp
multicastclient	configuration.cpp	Implements the operations of the Configuration class
multicastclient	configuration.h	Function prototype for configuration.cpp
multicastclient	digitalpstnboard.cpp	Implements the operations of the DigitalPSTNBoard class
multicastclient	digitalpstnboard.h	Function prototype for digitalpstnboard.cpp
multicastclient	digitalpstndevice.cpp	Implements the operations of the DigitalPSTNDevice class
multicastclient	digitalpstndevice.h	Function prototype for digitalpstndevice.cpp
multicastclient	gwcall.cpp	Implements the operations of the GWCall class
multicastclient	gwcall.h	Function prototype for gwcall.cpp
multicastclient	incfile.h	Function prototype for Global Call and R4 functions

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

multicastclient	ipmediaboard.cpp	Implements the operations of the IPBoard class
multicastclient	ipmediaboard.h	Function prototype for ipboard.cpp
multicastclient	ipmediadevice.cpp	Implements the operations of the IPDevice class
multicastclient	ipmediadevice.h	Function prototype for ipdevice.cpp
multicastclient	main.cpp	Contains the main function and the Wait for Key
multicastclient	main.h	Function prototype for main.cpp
multicastclient	multicastclient.ver	Demo version information
multicastclient	pstncallcontrol.cpp	Implements the operations of the PstnCallControl class
multicastclient	pstncallcontrol.h	Function prototype for pstncallcontrol.cpp
multicastclient	r4device.cpp	Implements the operations of the R4Device class
multicastclient	r4device.h	Function prototype for r4device.cpp
multicastclient	r4logicalboard.cpp	Implements the operations of the R4LogicalBoard class
multicastclient	r4logicalboard.h	Function prototype for r4logicalboard.cpp
multicastclient	resourcemanager.cpp	Implements the operations of the ResourceManager class
multicastclient	resourcemanager.h	Function prototype for resourcemanager.cpp
multicastclient	statistics.h	Call statistics defines
multicastclient	voiceboard.cpp	Implements the operations of the VoiceBoard class
multicastclient	voiceboard.h	Function prototype for voiceboard.cpp
multicastclient	voiceresource.cpp	Implements the operations of the VoiceResource class
multicastclient	voiceresource.h	Function prototype for voiceresource.cpp
multicastclient (Windows only)	multicastclient.dsp	Visual C++ project file
multicastclient (Windows only)	multicastclient.dsw	Visual C++ project workspace
multicastclient (Windows only)	multicastclient.rc	Resource file
multicastclient (Windows only)	resource.h	Microsoft Developer Studio generated include file used by multicastclient_r4.rc
multicastclient\ release (Windows only)	multicastclient.cfg	Demo configuration file
multicastclient\ release (Windows only)	multicastclient.exe	Demo executable

5.1.2 Utility Files

The utility files listed in Table 4 are located in:

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

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

Directory	File Name	Purpose
utilcpp	utilcpp.ver	Utility library version information
utilcpp	log.cpp	Debugging functions
utilcpp	log.h	Function prototype for log.cpp
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 Client (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 Client (IPML) demo classes. It contains the following information:

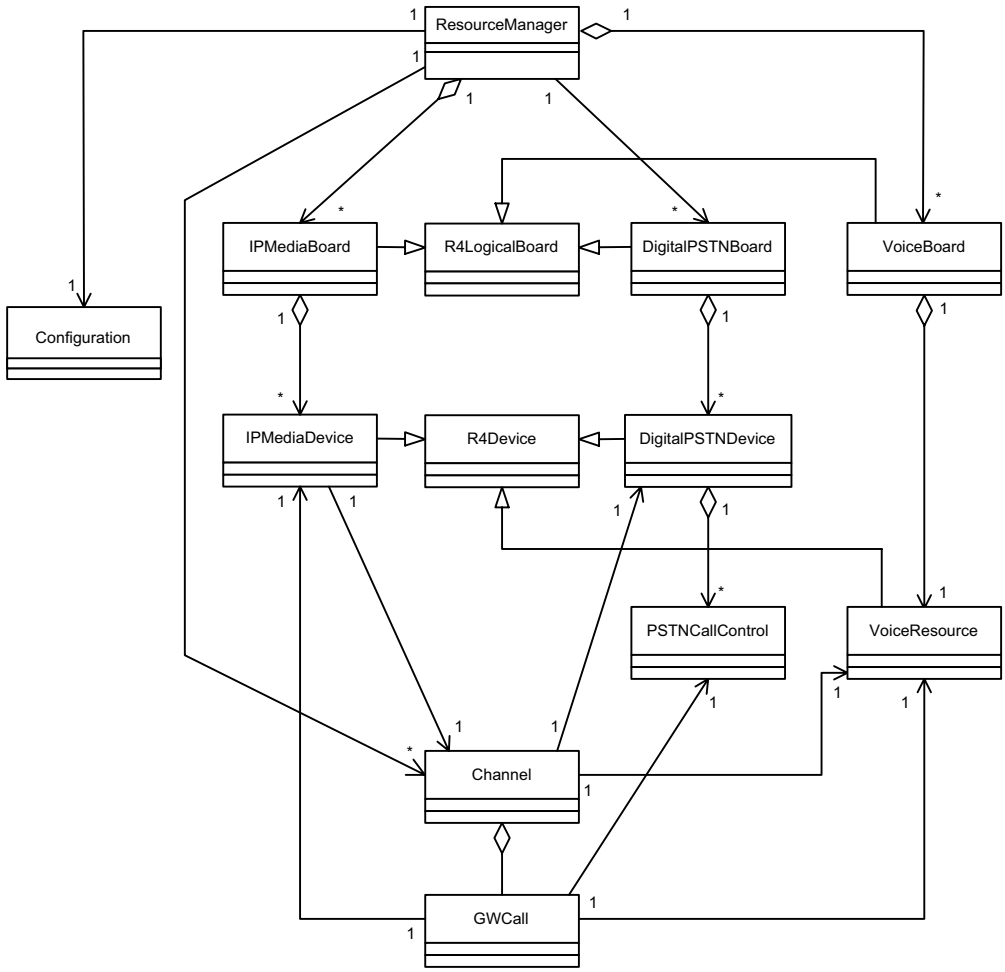
- Class Diagram
- Channel Class
- Configuration Class
- DigitalPSTNBoard Class
- DigitalPSTNDevice Class
- GWCall Class

- IPMediaBoard Class
- IPMediaDevice Class
- PSTNCallControl Class
- R4Device Class
- R4LogicalBoard Class
- ResourceManager Class
- VoiceBoard Class
- VoiceResource Class

5.2.1 Class Diagram

The following class diagram describes the relationship among the classes.

Figure 1. IP Multicast Client (IPML) Class Diagram



5.2.2 Channel Class

The main role of the Channel class is to control all resources related to a call. It contains all the resources related to a call and manages all gateway calls and its resources.

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

Table 6. Channel Class Attributes

Name	Access Privilege	Type	Description
m_pIPMediaDevice	public	IPMediaDevice*	IPMedia device of the channel
m_pdigPSTNDevice	public	DigitalPSTNDevice*	PSTN device of the channel
m_pVRDevice	public	VoiceResource*	The VoiceResource device of the channel
m_pLog	public	Log*	The log object of the channel
m_channelId	private	unsigned int	The channel identifier
m_GWCalls	private	GWCALL_LIST (an STL list of GWCall*)	List of the calls initiated on the channel
m_inService	private	bool	Indicates if the channel is in or out of service

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_pstnProtocol	public	char	The PSTN protocol to be used. It is read from the configuration file and updated during initialization. It does not change until demo termination. It is accessed while opening the PSTNBoard and DigitalPSTN devices.
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, read whenever printing to the log. The log level is the same for all the log objects.

Table 7. Configuration Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_QoSFlag	public	int	Set to “1” if the QoS feature is enabled by the -q command line option and accessed before calling any QoS function. It is read from the configuration file during initializations and is not changed until demo termination.
m_userChannels	public	unsigned int	Indicates the number of channels that the demo will work with, updated after reading the -n command line option. It remains unchanged until demo termination and is read in order to calculate the minimum number of channels.
m_chanOfServer	public	ChannelOfServer[ ]	Array that contains all the channel information from the configuration file, such as coder info and the log file flag.
m_stage	private	unsigned char	The stage of parsing the configuration file
m_line	private	int	The line currently being parsed in the configuration file
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_cfgFile	private	char*	The configuration file name
m_logFileFlag	private	int	Flag for using log files, one for each channel (by the -l command line option)

5.2.4 DigitalPSTNBoard Class

The main role of the DigitalPSTNBoard class is to initiate the digital PSTN boards and manage the digital PSTN device database. The DigitalPSTNBoard class contains all the digital PSTN devices available in the system.

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

Table 8. DigitalPSTNBoard Class Attributes

Name	Access Privilege	Type	Description
m_pstnDevices	public	DigitalPSTNDevice (MAX_BOARD_DEVICES)	Array of PSTN devices found on the PSTN board. Accessed while looking for a free DigitalPSTNDevice in order to allocate it to a Channel.

5.2.5 DigitalPSTNDevice Class

The main role of the DigitalPSTNDevice class is to provide R4 functionality to a digital PSTN device. It reflects the device status and manages all calls related to that device. The DigitalPSTNDevice class represents any digital PSTN R4 device.

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

Table 9. DigitalPSTNDevice Class Attributes

Name	Access Privilege	Type	Description
m_lineDevice	public	LINEDEV	Line device of the PSTN device - valid after opening the PSTN device
m_PSTNCCs	public	PSTNCallControl[ ]	Array that holds the PSTN device call control
m_localPhoneNumber	public	char	Local phone number used by the **makeCall( )** function. Read from the configuration file.

5.2.6 GWCall Class

The main role of the GWCall class is to control all resources related to a call. It contains all the resources needed to establish a call:

- PSTNCallControl
- IPMediaDevice
- VoiceResource

The GWCall class reflects the call state to the other classes.

The GWCall class attributes are described in Table 10. Refer to the source code for method information.

Table 10. GWCall Class Attributes

Name	Access Privilege	Type	Description
m_pIPMediaDevice	public	IPMediaDevice*	The IPMedia device of the call
m_pPSTNCC	public	PSTNCallControl*	PSTN call control object of the call, handling the PSTN call control
m_pVRDevice	public	VoiceResource*	The VoiceResource device of the call
m_pLog	public	Log*	Log object of the call

Table 10. GWCall Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_currentState	private	E_StateMachine	Current channel state
m_DropFlag	private	int	Binary Flag. First bit is "1" if, and only if, the **gc_dropCall( )** function was called to drop the PSTN call. The second bit is the same, but for the IP side.

5.2.7 IPMediaBoard Class

The main role of the IPMediaBoard class is to initiate the IPMedia board. It manages the IP Media device database, containing all the IP Media devices available to the system, and reflects the IP Media device repository.

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

Table 11. IPMediaBoard Class Attributes

Name	Access Privilege	Type	Description
m_ipMediaDevices	public	IPMediaDevice [MAX_BOARD_DEVICES]	Array of the IPMedia devices found on the IP board, accessed while looking for a free IPMedia device in order to allocate it for a Channel.

5.2.8 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 12. Refer to the source code for method information.

Table 12. IPMediaDevice Class Attributes

Name	Access Privilege	Type	Description
m_QoSStatus	public	bool	True if, and only if, the QoS feature is enabled. It is updated after reading the -q command line option and never changed until demo termination.
m_QoSAlarmFile	public	FILE*	Pointer to the QoS statistics file
m_QoSAlarmFileName	public	Char*	The QoS statistics file name - used when opening the file
m_isFree	public	bool	Indicates if the IPMedia device is free

Table 12. IPMediaDevice Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_channelOfServer	public	int	The multicast server channel
m_currentState	private	E_StateMachine	The current state of the channel
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_localMediaInfo	private	IPM_MEDIA_INFO	The local media information
m_pQoSThresholdInfo	private	PIPM_QOS_ THRESHOLD_INFO	The QoS thresholds, updated after reading the configuration file. Used to set the media device QoS thresholds.
m_SessionInfo	private	IPM_SESSION_INFO	Used to get the session information before stopping the session. It's address is sent to the **ipm_GetSessionInfo( )** function.
m_QoSInfo	private	QoSInfo	The QoS information - threshold and the reset alarm state flag, read from the configuration file.
m_rxCodec	private	IPM_CODER_INFO	The Rx coder information read from the configuration file.
m_txCodec	private	IPM_CODER_INFO	The Tx coder information read from the configuration file.
m_portInfo	private	IPM_PORT_INFO	The multicast port and ip address read from the configuration file, used in the **ipm_StartMedia( )** function.
m_remoteMediaInfo	private	IPM_MEDIA_INFO	The remote GW media information

5.2.9 PSTNCallControl Class

The main role of the PSTNCallControl class is to provide a Global Call functionality interface to manage a call. It represents one PSTN call on a specific digital PSTN R4 device and reflects the call status to the other classes.

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

Table 13. PSTNCallControl Class Attributes

Name	Access Privilege	Type	Description
m_isAllocated	public	int	TRUE if the CC object is allocated to a call
m_pLog	public	Log*	The log object used to control the CC object printouts
m_lineDevice	public	LINEDEV	Line device of the PSTN device that the PSTNCC belongs to
m_currentState	private	E_StateMachine	Current state that the PSTN call control object found in the state machine
m_crn	private	CRN	The call reference number of the PSTN call control object

5.2.10 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 14.

Table 14. R4Device Class Attributes

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

5.2.11 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 15. Refer to the source code for method information.

Table 15. R4LogicalBoard Class Attributes

Name	Access Privilege	Type	Description
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
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

5.2.12 ResourceManager Class

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

- all system channels
- configuration object
- maps R4 device handles to channels
- boards from all types

The ResourceManager class attributes are described in Table 16. Refer to the source code for method information.

Table 16. ResourceManager Class Attributes

Name	Access Privilege	Type	Description
m_PSTNBoards	public	DigitalPSTNBoard	Array that includes the PSTN boards available in the system, filled while initializing the PSTN boards that had been previously found. Not changed until the demo termination. This array is read whenever the system wants to get a free PSTN device
m_IPBoards	public	IPMediaBoard*	Array that includes the IP boards available in the system
m_voiceBoards	public	VoiceBoard*	Array that includes the voice boards available in the system

Table 16. ResourceManager Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_ptheConfiguration	public	static Configuration*	An instance of the Configuration class used to determine the configuration of the system during initialization. This object is constructed before the initialization and is changed during the initialization while getting data from the configuration file and command line options. There is one instance of this object in the system because there is only one configuration file. This object is accessed during the initialization only, and after that all data related to the different objects is passed to them and the configuration object is destructed.
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). This integer is determined after detecting the system devices and getting the number of channels that the user wants and is not changed until the demo termination.
DeviceHandleToChannel	public	static unsigned int	Maps the devices to channels through their handles (gotten after opening each channel) to enable handling the SRL events. The table is filled in when opening each device.
m_systemChannels	public	static Channel*	Array that contains all the channels used by the application. This array is filled during the initialization, after the minimum number of channels is known. The array is accessed whenever the application wants to perform an action on a channel.
m_numOfPSTNBoards	public	int	The number of PSTN boards in the system. This integer is updated after detecting the PSTN boards and does not change until demo termination.
m_numOfVoiceBoards	public	int	The number of Voice boards in the system. This integer is updated after detecting the Voice boards and does not change until demo termination.

Table 16. ResourceManager Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_numOfIPBoards	public	int	The number of IP boards in the system. This integer is updated after detecting the IP boards and does not change until demo termination.
m_pLog	public	static Log*	A log instance used during initialization. All the printouts are sent to the monitor, after which it is destructed. This instance is necessary because during the initialization, there are no channel instances and therefore no log instances. In order to see logs during initialization, the application creates a global log instance for all the devices during initialization and kills it after creating the channel objects and attributing the devices to channels.

5.2.13 VoiceBoard Class

The main role of the VoiceBoard class is to initiate the Voice board. It contains all the voice resources available to the system and reflects the Voice Resources database.

The VoiceBoard class attributes are described in Table 17. Refer to the source code for method information.

Table 17. VoiceBoard Class Attributes

Name	Access Privilege	Type	Description
m_voiceDevices	public	VoiceDevice [MAX_BOARD_DEVICES]	Array of the voice devices found on the voice board, accessed while looking for a free Voice device in order to allocate it for a Channel.

5.2.14 VoiceResource Class

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

The VoiceResource class attributes are described in Table 18.

Table 18. VoiceResourceClass Attributes

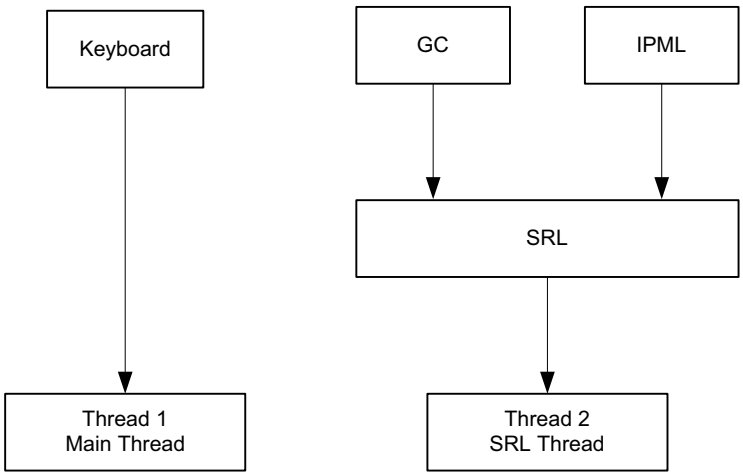
Name	Access Privilege	Type	Description
m_digitsBuffer	public	DV_DIGIT	Contains the digits that were gotten when **getDigits( )** was called.
m_channelOfServer	public	int	The multicast server channel

5.3 Threads

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

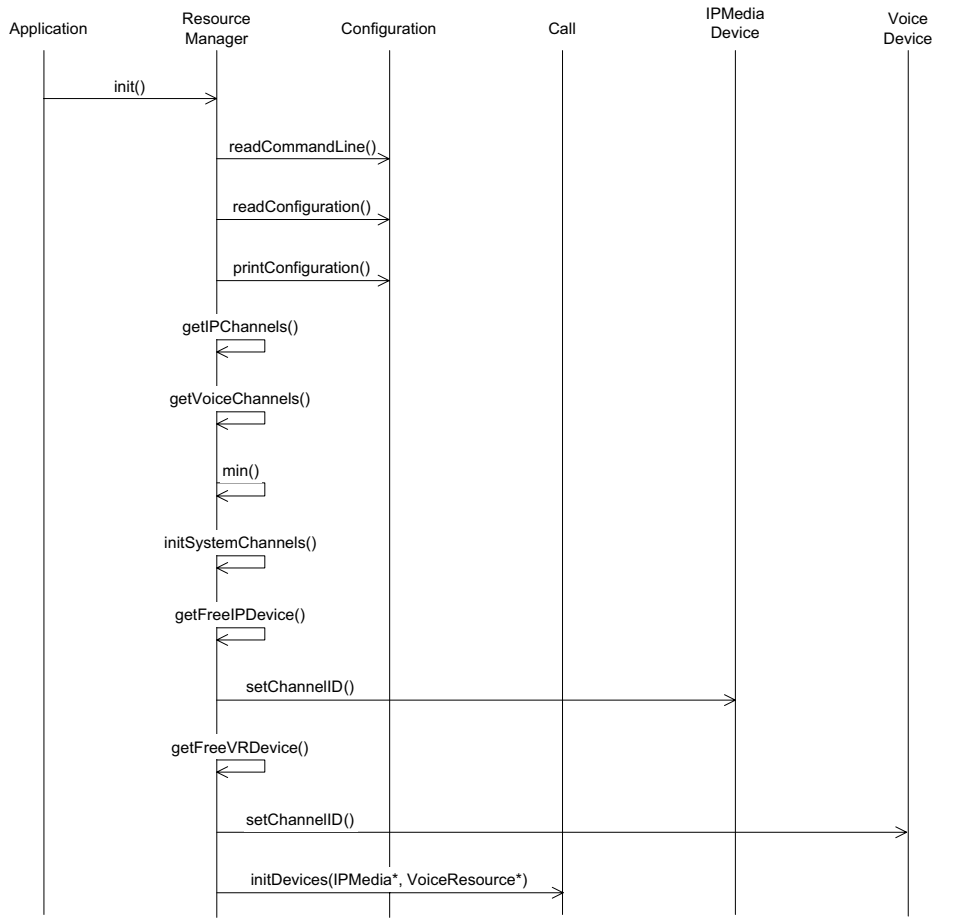
- The first thread (main) is created by the demo application to get the keyboard input
- The second thread is an SRL thread, created as a result of the demo application calling **sr_enblhdlr( )** in Windows. In Linux, the thread must be explicitly created.

Figure 2. Thread Diagram



5.4 Initialization

Figure 3. IP Multicast Client (IPML) System Initialization



This section describes the demo initialization as shown in Figure 3.

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 36.
2. The application **main( )** function then calls **resourceManager.init( )**, which does the following:
   a. Reads the command line options
   b. Reads and parses 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 Client (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 Client (IPML) state machines. It contains the following topics:

6.1 GWCall State Machine

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

- GWCall State Machine Description
- GWCall::gateNull State
- GWCall::gateOfferingFromPSTN State
- GWCall::gatePstnConnected State
- GWCall::gateStartingMedia State
- GWCall::gateConnected State
- GWCall::gateDropping State
- GWCall::gateDropping State

6.1.1 GWCall State Machine Description

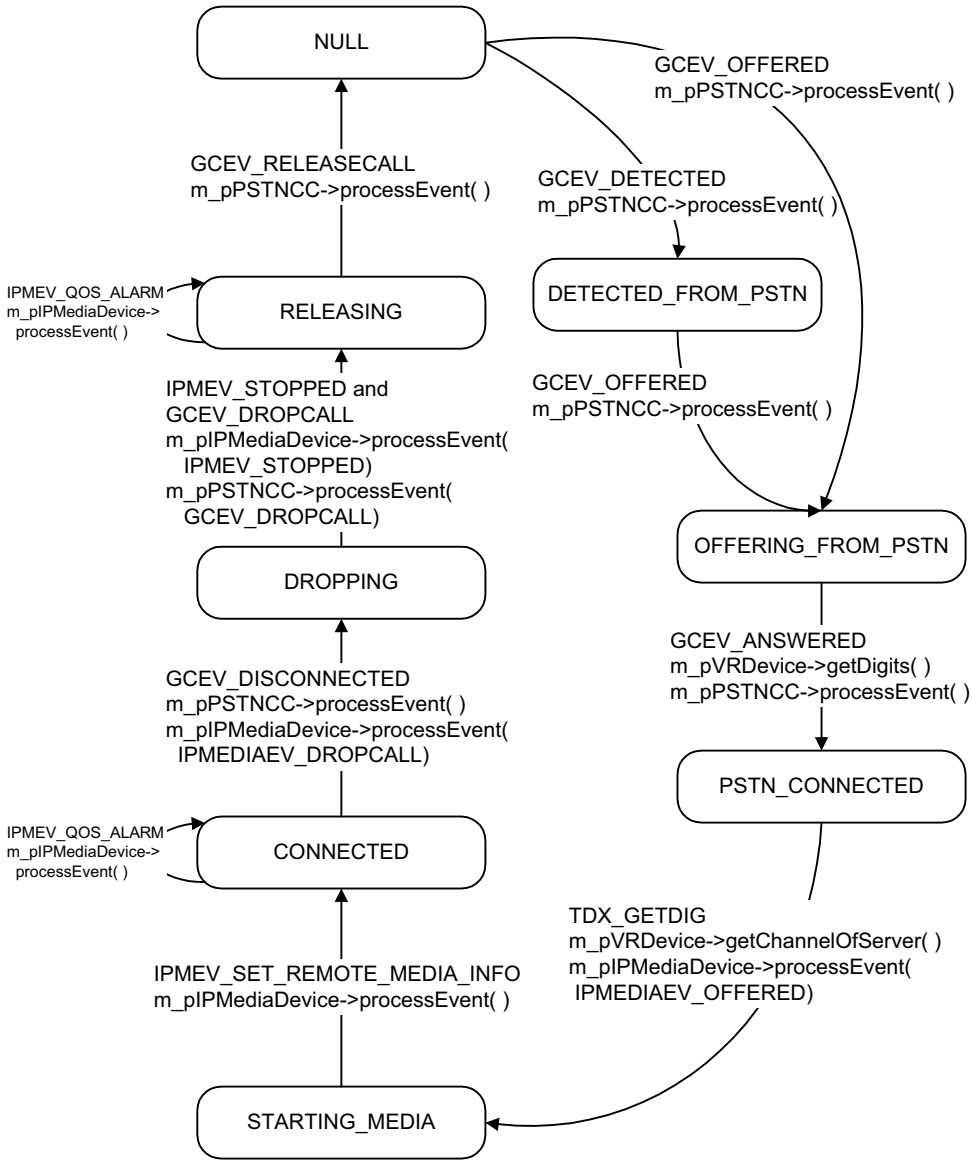
All channels are initialized to the NULL state upon application start.

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 GWCall class.

Figure 4. GWCall State Machine



6.1.2 GWCall::gateNull State

The application waits in the gateNull state for either a GCEV_OFFERED or a GCEV_DETECTED event.

In the case of GCEV_OFFERED, the application calls **processEvent(GCEV_OFFERED)** from the PSTNCC module and the call state transitions to gateOfferringFromPSTN. See Section 6.3, "PSTNCallControl State Machine", on page 42 for a description of the PSTNCC module state machine.

In the case of GCEV_DETECTED, the application calls **processEvent(GCEV_DETECTED)** from the PSTNCC module and the call state transitions to gateDetectedFromPSTN.

If there is no IPMedia device available, the application calls **processEvent(CCEV_DROPCALL)** from the PSTNCC module and the call state transitions to gateDropping.

6.1.3 GWCall::gateDetectedFromPSTN State

The application waits for a GCEV_OFFERED event. Upon receipt of the event it calls **processEvent(GCEV_OFFERED)** from the PSTNCC module and the call state transitions to gateOfferingFromPSTN.

6.1.4 GWCall::gateOfferingFromPSTN State

The application waits for a GCEV_ANSWERED event. Upon receipt of the event, it calls **getDigits( )** from the VoiceResource module to collect the DTMF digits from the voice resource channel digit buffer and calls **processEvent(GCEV_ANSWERED)** from the PSTNCC module. The call state transitions to gatePstnConnected.

If the application receives either a GCEV_DISCONNECTED or GCEV_TASKFAIL event, it calls **processEvent(CCEV_DROPCALL)** from the PSTNCC module and the call state transitions to gateDropping.

6.1.5 GWCall::gatePstnConnected State

The application waits for a TDX_GETDIG event. Upon receipt of the event, it calls **getChannelOfServer( )** from the VoiceResource module to get the server channel from the digit buffer. It sets m_channelOfServer in the IPMediaDevice to the same value and calls **processEvent(IPMEDIAEV_OFFERED)** from the IPMediaDevice module. See Section 6.2, “IPMediaDevice State Machine”, on page 40 for a description of the IPMediaDevice module state machine. The call state transitions to gateStartingMedia.

If the application receives a GCEV_DISCONNECTED event, it calls **processEvent(GCEV_DISCONNECTED)** from the PSTNCC module and the call state transitions to gateDropping.

6.1.6 GWCall::gateStartingMedia State

The application waits for a IPMEV_SET_REMOTE_MEDIA_INFO event. Upon receipt of the event, it calls **processEvent(IPMEV_SET_REMOTE_MEDIA_INFO)** from the IPMediaDevice module and the call state transitions to gateConnected.

If the application receives a GCEV_DISCONNECTED or IPMEV_ERROR event, it calls **processEvent(CCEV_DROPCALL)** from the PSTNCC module and **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and the call state transitions to gateDropping.

6.1.7 GWCall::gateConnected State

The application waits for a GCEV_DISCONNECTED event. Upon receipt of the event, it calls **processEvent(GCEV_DISCONNECTED)** from the PSTNCC module and **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module. The call state transitions to gateDropping.

If the application receives a IPMEV_QOS_ALARM event, it calls **processEvent(IPMEV_QOS_ALARM)** from the IPMediaDevice module. The call state remains in gateConnected.

6.1.8 GWCall::gateDropping State

The application waits for both an IPMEV_STOPPED and GCEV_DROPCALL event. Upon receipt of both events, it calls **processEvent(IPMEV_STOPPED)** from the IPMediaDevice module and **processEvent(GCEV_DROPCALL)** from the PSTNCC module. The call state transitions to gateReleasing.

If the application receives a IPMEV_QOS_ALARM event, it calls **processEvent(IPMEV_QOS_ALARM)** from the IPMediaDevice module. The call state remains in gateDropping.

6.1.9 GWCall::gateReleasing State

The application waits for a GCEV_RELEASECALL event. Upon receipt of the event, it calls **processEvent(GCEV_RELEASECALL)** from the PSTNCC module and the call state transitions to gateNull.

6.2 IPMediaDevice State Machine

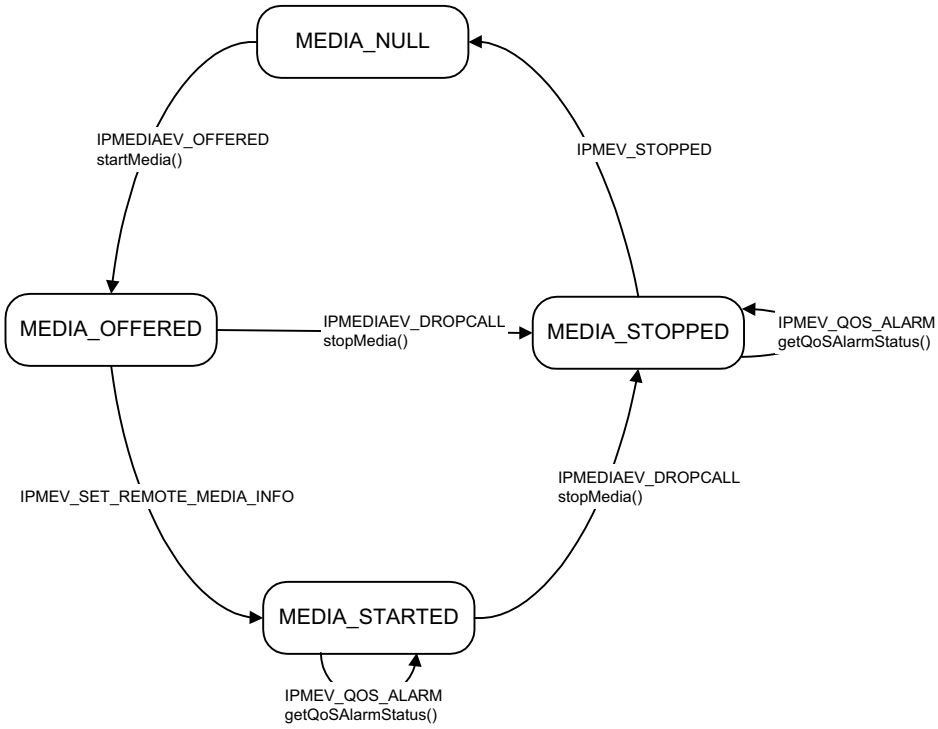
This section describes the IPMediaDevice state machine. It contains the following topics:

- IPMediaDevice State Machine Description
- IPMediaDevice::mediaNull State
- IPMediaDevice::mediaOffered State
- IPMediaDevice::mediaStarted State
- IPMediaDevice::mediaStopped State

6.2.1 IPMediaDevice State Machine Description

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

Figure 5. IPMediaDevice State Machine



6.2.2 IPMediaDevice::mediaNull State

The application waits for a IPMEDIAEV_OFFERED event in the MEDIA_NULL_STATE. Upon receiving this event it calls **startMedia( )**. The state transitions to mediaOffered.

If the application receives IPMEDEV_DROPCALL, it ignores the event and remains in the mediaNull state.

6.2.3 IPMediaDevice::mediaOffered State

The application waits for a IPMEV_SET_REMOTE_MEDIA_INFO event. Upon receipt of the event the call state transitions to mediaStarted.

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

6.2.4 IPMediaDevice::mediaStarted State

The application waits for an IPMEDIAEV_DROPCALL event. Upon receipt of this event, it calls **stopMedia( )** and the state transitions to mediaStopped. If QoS is enabled, the application calls **getSessionInfo( )** before calling **stopMedia( )**.

If the application receives a IPMEV_QOS_ALARM event, it calls **getQoSAlarmStatus( )**. The call state remains in mediaStarted.

6.2.5 IPMediaDevice::mediaStopped State

The application waits for an IPMEV_STOPPED event. Upon receipt of this event, the state transitions to mediaNull.

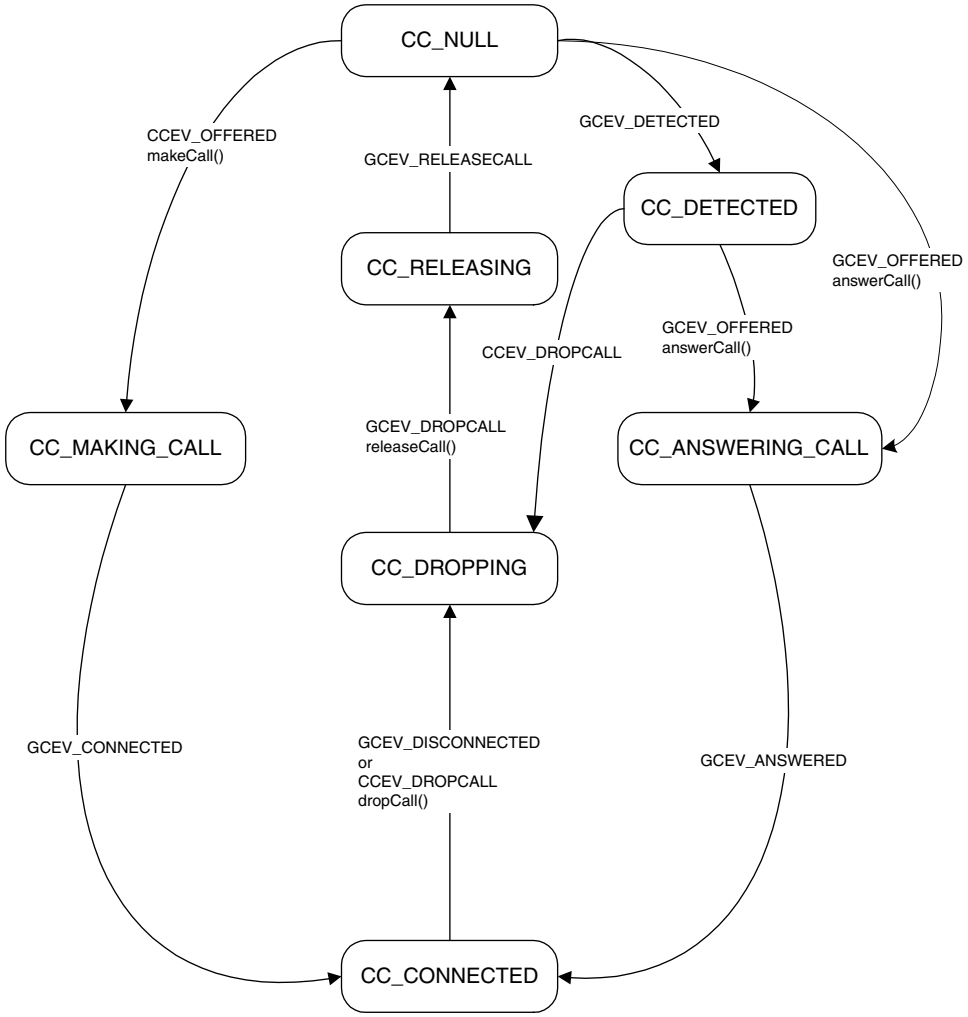
6.3 PSTNCallControl State Machine

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

6.3.1 PSTNCallControl State Machine Description

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

Figure 6. PSTNCallControl State Machine



6.3.2 PSTNCallControl::CCNull State

The application waits for an offered event. If it receives GCEV_OFFERED, the application calls **answerCall( )** and the call state transitions to CCAnsweringCall. If it receives CCEV_OFFERED, the application calls **makeCall( )** and the call state transitions to CCMakingCall.

The application can also receive a GCEV_DETECTED event. In this case, the call state transitions to CCDetected.

If the application receives a CCEV_DROPCALL, it calls **dropCall( )** and the call state transitions to CCDropping.

6.3.3 PSTNCallControl::CCDetected State

The application waits for a GCEV_OFFERED event. Upon receipt of the event, the application calls **answerCall( )** and the call state transitions to CCAnsweringCall.

6.3.4 PSTNCallControl::CCAnsweringCall State

The application waits for a GCEV_ANSWERED event. Upon receipt of the event, the call state transitions to CCConnected.

If the application receives a GCEV_DISCONNECTED event, it calls **dropCall( )** and the call state transitions to CCDropping.

6.3.5 PSTNCallControl::CCMakingCall State

The application waits for a GCEV_CONNECTED event. Upon receipt of the event, the call state transitions to CCConnected.

If the application receives a CCEV_DROPCALL event, it calls **dropCall( )** and the call state transitions to CCDropping.

6.3.6 PSTNCallControl::CCConnected State

The application waits for either a GCEV_DISCONNECTED or a CCEV_DROPCALL event. Upon receipt of one of these events, it calls **dropCall( )** and the call state transitions to CCDropping.

6.3.7 PSTNCallControl::CCDropping State

The application waits for a GCEV_DROPCALL event. Upon receipt of the event, it calls **releaseCall( )** and the call state transitions to CCReleasing.

6.3.8 PSTNCallControl::CCReleasing State

The application waits for a GCEV_RELEASECALL event. Upon receipt of the event, the channel is deallocated and the call state transitions to CCNull.

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