IP Media Gateway (IPML)

Demo Guide

September 2002

05-1823-001

Publication Date: September 2002

Document Number: 05-1823-001

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 Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices** and other contact information, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Contents

Figures

Tables

About This Publication

The following topics provide information about this guide:

Purpose

This guide provides information on the IP Media Gateway (IPML) demo that is available with Intel® Dialogic® System Release 6.0 for the Linux* and Windows* Operating Systems on Intel® Architecture. This guide describes the demo, its requirements, and details on how it works.

Intended Audience

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:

- *System Release 6.0 Release Update* for information on problems fixed, known problems and workarounds, compatibility issues and last minute updates not documented in the published information.
- *DM3 for Linux Configuration Guide*
- *Intel® NetStructure™ on DM3 Architecture for cPCI on Windows Configuration Guide*
- *Intel® NetStructure™ IPT Series for Linux Configuration Guide*
- *Intel® NetStructure™ IPT Series on Windows Configuration Guide*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/network/csp/* (for product information)

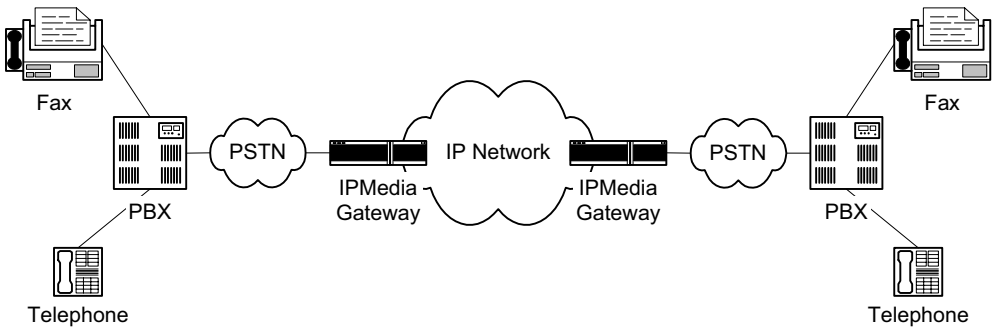
Demo Description 1

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

The IP Media (IPML) Demo demonstrates how the IP Media Library (IPML) API may be used to build a PSTN-IP gateway when using a proprietary host-based protocol stack, i.e., a protocol stack not supplied with the system release. The demo source code can be used as sample code for those who want to begin developing an application from a working application.

A basic topology is presented in Figure 1.

Figure 1. IP Media (IPML) Topology



Note: This guide does not discuss the proprietary protocol stack design. Review the source code to better understand how the stack is implemented.

The IP Media (IPML) demo supports the following features:

- Configuration file
- Command line options
- Output log files
- Printing to the monitor
- QoS
- Accepts IP calls
- Places IP calls

The following features are not supported by this demo application:

- Analog PSTN interface
- UII message
- NonStdCmd message
- NonStdParm data

- Q.931Facility message

The IP Media (IPML) demo is a cross-OS demo, running under the Windows and 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, 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 Media (IPML) demo. It contains the following topics:.

2.1 Hardware Requirements

This demo requires two separate gateways in order to create an end-to-end voice path, due to the nature of the proprietary call control written for the demo.

To run the IP Media (IPML) demo, you need, for each gateway:

- One of the following:
  - Intel® NetStructure™ DMIP Series board
  - Intel® NetStructure™ IPT Series board
    - also requires an Intel® NetStructure™ DM/V-A series board for PSTN connection
- IP network cable

For other hardware requirements, such as memory requirements, see the *Release Guide* for your system release.

2.2 Software Requirements

To run the IP Media (IPML) demo, you need the Intel® Dialogic® System Release 6.0 for the Linux and Windows Operating Systems on Intel Architecture. For a list of operating system requirements see the *Release Guide* for your system release.

See Section 3.3, “Compiling and Linking”, on page 20 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 Media (IPML) demo. It provides information about the following topics:

3.1 Connecting to External Equipment

The IP Media (IPML) demo uses a proprietary call control stack that is not compatible with other stacks or IP clients (such as Windows* NetMeeting*). The gateway must be connected to a second gateway also running the IP Media (IPML) application in order to open an end-to-end voice path.

3.2 Editing Configuration Files

This section discusses how to configure the demo for your system. It contains the following topics:

- File Location
- Editing the ipmedia_r4.cfg Configuration File

3.2.1 File Location

Before running the IP Media (IPML) demo, modify the *ipmedia_r4.cfg* file to reflect your system environment. Use a text editor and open the file from:

- Windows: *C:\Program Files\dialogic\demos\gc_demos\ipdemos\ipmedia_r4\release*
- Linux: */usr/dialogic/demos/ipt/gc_demos/ipmedia_r4/*

3.2.2 Editing the ipmedia_r4.cfg Configuration File

Below is an example of the *ipmedia_r4.cfg* file. Update the following information:

Destination host IP
: IP address of the NIC on the destination GW host

Destination host port
: IP port of the destination GW host

Local host port
: IP port of the local host

first call id
: Demo proprietary call ID. Each gateway must have a unique set of call IDs. This information is transmitted during call negotiation. This value is the first value in the range of call IDs for the gateway.

last call id
: The last value in the range of call IDs for the gateway.

PSTN protocol
: The PSTN protocol supported by the gateway. Possible values are: T1, E1, ISDN (including NFAS), CAS.

DTMF mode
: Specifies how DTMF tones are transmitted. Possible values are: RTPInBand (usually used with G.711 coders), OutOfBand (usually used with low bandwidth coders, e.g., GSM), RTPRFC2833.

VoiceRxCodecs
: Describes the receive voice coder. The parameters are as follows:
  - CoderType – The type of coder. See the *System Release Update* for specific information about coder support in this release.
  - CoderFramesPerPkt – Specify the number of frames per packet for the selected coder. See the *System Release Update* for specific information about coder support in this release.
  - CoderFrameSize – Specify the frame size for the selected coder. See the *System Release Update* for specific information about coder support in this release.
  - CoderVAD – Specify if VAD is active. See the *System Release Update* for specific information about coder support in this release.
  - Payload – Describes the static payload type values for the PT field of the RTP data header as described in RFC 1890.
  - RedPayload – 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.
  - RFC2833Payload – Describes the static payload type values when sending DTMF according to RFC2833.

VoiceTxCodecs
: Describes the transmit voice coder. See VoiceRxCodecs for a description of the parameters.

Data Codecs
: Describes the fax coder parameters. See VoiceRxCodecs for a description of the parameters.

Quality of Service
: The application can set threshold values to monitor the quality of service during calls. A fault occurs when the result of a measurement of a QoS parameter crossed a predefined threshold. A success occurs when the result of a measurement of a QoS parameter did not cross a predefined threshold. The QoS parameters are measured during time intervals, starting when a call is established. The following parameters are supported:
  - MediaAlarmLostPackets – indicates that the percentage of packets lost during a call exceeded its threshold value
  - MediaAlarmJitter – indicates that the jitter (as defined in RFC 1889) exceeded its threshold value

QoS Attributes
: Each parameter has six attributes:

- Threshold – defines when a QoS parameter is in a fault condition. A fault occurs when the result of a measurement of a QoS parameter crossed the Threshold value.
- DebounceOn – the time during which faults are measured (in msec., must be multiple of Interval)
- DebounceOff – the time during which successes are measured (in msec., must be multiple of Interval)
- Interval – the amount of time between two QoS parameter measurements (in multiples of 100 msec)
- Percent_Fail – the threshold of failures during the DebounceOn time (expressed as a percentage of failures)
- Percent_Success – the threshold of successes during the DebounceOn time (expressed as a percentage of successes)

The default values are as follows:

	Threshold	DebounceOn	DebounceOff	Interval	Percent_ Fail	Percent_ Success
Lost packets	20	10000	10000	1000	60	40
Jitter	60	20000	60000	5000	60	40

DigitMap

Sets the number of digits to be transmitted when sending DTMF

Sample Configuration File

```
Destination host IP = 10.242.214.20
Destination host port = 3000
Local host port = 3000
first call id = 1
last call id = 999

# PSTN protocol - possible values: T1 / E1, ISDN (including NFAS), CAS.
PSTN protocol = T1 isdn

Channel = 1 - 30
{
# DTMF modes  - possible values: RTPInBand, OutOfBand, RTPRFC2833.
DTMF mode = rtpinband
VoiceRxCodecs
{
CoderType = g711mulaw
CoderFramesPerPkt = 1
CoderFrameSize = 30
CoderVAD = 0
Payload = 0
RedPayload = 0
RFC2833Payload = 100
}

VoiceTxCodecs
{
CoderType = g711mulaw
CoderFramesPerPkt = 1
CoderFrameSize = 30
CoderVAD = 0
Payload = 0
RedPayload = 0
RFC2833Payload = 100
}
```

```
DataCodecs
{
CoderType = g711mulaw
CoderFramesPerPkt = 1
CoderFrameSize = 30
CoderVAD = 0
Payload = 0
RedPayload = 0
}
#QoS
   LostPackets_Threshold       = 20    # Threshold value
   LostPackets_DebounceOn      = 10000 # Threshold debounce ON
   LostPackets_DebounceOff     = 10000 # Threshold debounce OFF
   LostPackets_Interval        = 1000  # Threshold Time Interval (ms)
   LostPackets_PercentSuccess  = 60    # Threshold Success Percent
   LostPackets_PercentFail     = 40    # Threshold Fail Percent
   Jitter_Threshold            = 60    # Threshold value
   Jitter_DebounceOn           = 20000 # Threshold debounce ON
   Jitter_DebounceOff          = 60000 # Threshold debounce OFF
   Jitter_Interval             = 5000  # Threshold Time Interval (ms)
   Jitter_PercentSuccess       = 60    # Threshold Success Percent
   Jitter_PercentFail          = 40    # Threshold Fail Percent
   ResetAlarmState   = 0
}

#DigitMap = 1
#{
#NumOfDigits = 1
#}

#DigitMap = 2
#{
#NumOfDigits = 5
#}

#DigitMap = 3
#{
#NumOfDigits = 5
#}

#DigitMap = 4
#{
#NumOfDigits = 3
#}

#DigitMap = 5
#{
#NumOfDigits = 1
#}

#DigitMap = 6
#{
#NumOfDigits = 3
#}
```

3.3 Compiling and Linking

Compile the project within the following environments:

- Windows
  - Visual C++ environment, version 6

- Linux
  - g++

If you have added or changed files, to compile the project put the files in *dialogic\samples\gc_demos\ipdemos\ipmedia_r4*.

Set ipmedia_r4 as the active project and build in debug mode.

3.4 Selecting PCD/FCD Files

Choose a PCD and matching FCD file according to the PSTN protocol used.

Running the Demo 4

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

4.1 Starting the Demo

Windows

Select Run from the Start Menu. The demo executable file can be found in: *C:\Program Files\dialogic\demos\gc_demos\ipdemos\ipmedia_r4\release\ipmedia_r4.exe*. Click OK to run the IP Media (IPML) demo using the default settings.

Linux

The demo executable file can be found in: */usr/dialogic/demos/ipt/gc_demos/ipmedia_r4/ipmedia_r4*.

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.

Table 1. Command Line Switches

Switch	Action	Default
-n<*n*>	Sets the number of gateway channels	The lesser of Voice Devices or IP devices
-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
-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-TRACE – used at the start of the application entrance or the start of any function. • 4-INFO – prints data related to a specific action. **Note:** Debug level is inclusive; higher levels include all lower levels	-d0 (Fatal)
-q	Activates Quality of Service	Disabled
-c <filename>	Configuration file name	-c *ipmedia_r4.exe*
-h/?	Prints the command syntax to the screen	Off

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 commands listed in Table 2:

Table 2. Runtime Keyboard Commands

Command	Function
c or C	Prints channel statistics to file (statistics.log)
d<n> or D<n>	Change debug level during runtime, where <n> is the debug level
f<*value*> or F<*value*>	Change fax mode. Possible values are: • t or T for T.38 • g or G for G.711
m or M	Print log files for up to 2 channels to the screen
q or Q or Ctrl+c	Terminates the application
r or R	Send RFC2833 messages

4.4 Stopping the Demo

The IP Media (IPML) demo runs until it is terminated. Press “q” or “Q” or “Ctrl+c” to terminate the demo application.

Demo Details 5

This chapter discusses the IP Media (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

In Windows the source code files listed in Table 3 are located in:
C:\Program Files\dialogic\demos\gc_demos\ipdemos\ipmedia_r4.

In Linux the source code files listed in Table 3 are located in:
/usr/dialogic/demos/ipt/gc_demos/ipmedia_r4/.

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

Directory	File Name	Purpose
ipmedia_r4	channel.cpp	Implements the operations of the Channel class
ipmedia_r4	channel.h	Function prototype for channel.cpp
ipmedia_r4	configuration.cpp	Implements the operations of the Configuration class
ipmedia_r4	configuration.h	Function prototype for configuration.cpp
ipmedia_r4	pstnboard.cpp	Implements the operations of the DigitalPstnBoard class
ipmedia_r4	pstnboard.h	Function prototype for digitalpstnboard.cpp
ipmedia_r4	pstndevice.cpp	Implements the operations of the DigitalPstnDevice class
ipmedia_r4	pstndevice.h	Function prototype for digitalpstndevice.cpp
ipmedia_r4	gwcall.cpp	Implements the operations of the GWCall class
ipmedia_r4	gwcall.h	Function prototype for gwcall.cpp

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

Directory	File Name	Purpose
ipmedia_r4	incfile.h	Function prototype for Global Call and R4 functions
ipmedia_r4 (Linux only)	ipmedia_r4	Linux executable
ipmedia_r4 (Linux only)	ipmedia_r4.cfg	Demo configuration file
ipmedia_r4 (Windows only)	ipmedia_r4.dsp	Visual C++ project file
ipmedia_r4 (Windows only)	ipmedia_r4.dsw	Visual C++ project workspace
ipmedia_r4 (Windows only)	ipmedia_r4.rc	Resource file
ipmedia_r4	ipmedia_r4.ver	Demo version information
ipmedia_r4	main.cpp	Contains the main function and the Wait for Key
ipmedia_r4	main.h	Function prototype for main.cpp
ipmedia_r4 (Linux only)	makefile	Linux compilation file
ipmedia_r4	pstncallcontrol.cpp	Implements the operations of the PstnCallControl class
ipmedia_r4	psntcallcontrol.h	Function prototype for pstncallcontrol.cpp
ipmedia_r4 (Windows only)	resource.h	Microsoft Developer Studio generated include file used by ipmedia_r4.rc
ipmedia_r4	resourcemanager.cpp	Implements the operations of the ResourceManager class
ipmedia_r4	resourcemanager.h	Function prototype for resourcemanager.cpp
ipmedia_r4/release (Windows only)	ipmedia_r4.cfg	Demo configuration file
ipmedia_r4/release (Windows only)	ipmedia_r4.exe	Demo executable
ipmediamodule	ipmediaboard.cpp	Implements the operations of the IPMediaBoard class
ipmediamodule	ipmeadiaboard.h	Function prototype for ipmediaboard.cpp
ipmediamodule	ipmediadevice.cpp	Implements the operations of the IPMediaDevice class
ipmediamodule	ipmediadevice.h	Function prototype for ipmediadevice.cpp
basemodules	r4device.cpp	Implements the operations of the R4Device class
basemodules	r4device.h	Function prototype for r4device.cpp
basemodules	r4logicalboard.cpp	Implements the operations of the R4LogicalBoard class
basemodules	r4logicalboard.h	Function prototype for r4logicalboard.cpp
ipproprietary	client.cpp	Implementation of the client socket for sending messages over the IP
ipproprietary	client.h	Function prototype for client.cpp
ipproprietary	ipcallcontrol.cpp	Implements operations of the IPCallControl class

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

Directory	File Name	Purpose
ipproprietary	ipcallcontrol.h	Function prototype for ipcallcontrol.cpp
ipproprietary	ipmsg.cpp	Implements operations of the IPMsg class
ipproprietary	ipmsg.h	Function prototype for ipmsg.cpp
ipproprietary	ipprotocol.cpp	Implements operations of the IPProtocol class
ipproprietary	ipprotocol.h	Function prototype for ipprotocol.cpp
ipproprietary	ipprotocolmgr.cpp	Implements operations of the IPProtocolMgr class
ipproprietary	ipprotocolmgr.h	Function prototype for ipprotocolmgr.cpp
ipproprietary	server.cpp	Implementation of the server socket for receiving messages over IP
ipproprietary	server.h	Function prototype for server.cpp
ipproprietary	ipprotocoldefs.h	Global definitions
ipproprietary	ipproprietary.ver	Module library version information
ipproprietary (Windows only)	ipproprietary.dsp	Visual C++ project file
ipproprietary (Windows only)	ipproprietary.dsw	Visual C++ project workspace
ipproprietary\release (Windows only)	ipproprietary.lib	Compiled module library
ipproprietary (Linux only)	makefile.ipprotocol	Linux compilation file
ipproprietary (Linux only)	libipproprietary.a	Compiled module library

5.1.2 Utility Files

In Windows the utility files listed in Table 4 are located in:
C:\Program Files\dialogic\demos\gc_demos\ipdemos\ipmedia_r4.

In Linux the utility files listed in Table 4 are located in:
/usr/dialogic/demos/ipt/gc_demos/utilcpp/.

Table 4. Utility Files Used by the IP Media (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

Table 4. Utility Files Used by the IP Media (IPML) Demo (Continued)

Directory	File Name	Purpose
utilcpp (Windows only)	utilcpp.dsp	Utility library Visual C++ project file
utilcpp\release (Windows only)	utilcpp.lib	Compiled Utility library
/utilcpp (Linux only)	makefile.utilcpp	Compilation file
/utilcpp (Linux only)	libutilcpp.a	Compiled Utility library

5.1.3 PDL Files

In Windows the PDL files listed in Table 5 are located in:
C:\Program Files\dialogic\demos\gc_demos\ipdemos\pdl_win.

In Linux the PDL files listed in Table 6 are located in:
/usr/dialogic/demos/ipt/gc_demos/utilcpp/pdl_linux.

Table 5. PDL Files Used by the IP Media (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

Table 6. PDL Files Used by the IP Media (IPML) Demo - Linux OS

Directory	File Name	Purpose
pdl_linux	iptransport.cpp	PDL IP transport functions
pdl_linux	iptransport.h	Function prototype for iptransport.cpp
pdl_linux	libpdl.a	Compiled PDL library
pdl_linux	makefile.pdl	Compilation file
pdl_linux	pdl.c	Platform dependency functions
pdl_linux	pdl.h	Function prototype for pdl.c
pdl_linux	pdl.ver	PDL version information

5.2 Programming Model Classes

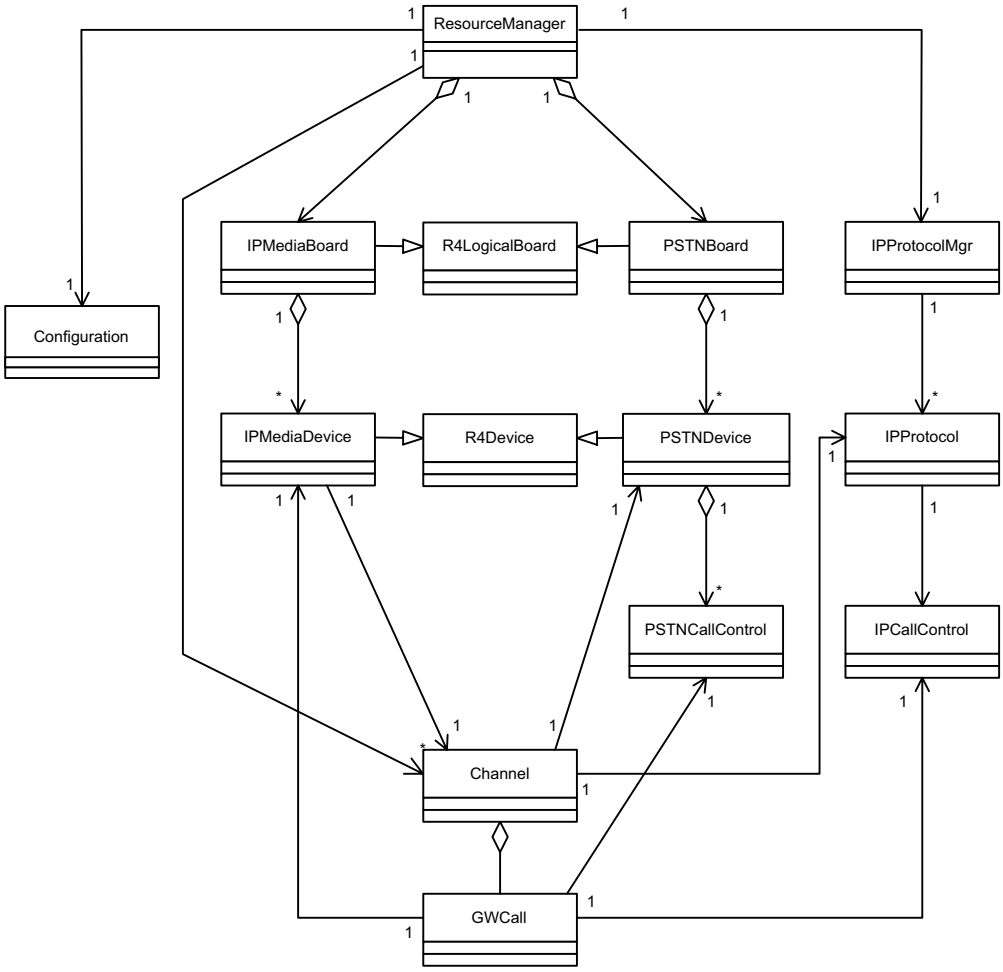
This section presents basic information about the IP Media (IPML) demo classes. It contains the following information:

- Class Diagram
- Channel Class
- Configuration Class
- DigitalPSTNBoard Class
- DigitalPSTNDevice Class
- GWCall Class
- IPCallControl Class
- IPMediaBoard Class
- IPMediaDevice Class
- IPMsg Class
- IPProtocol Class
- IPProtocolMgr Class
- PSTNCallControl Class
- R4Device Class
- R4LogicalBoard Class
- ResourceManager Class

5.2.1 Class Diagram

The following class diagram describes the relationship among the classes.

Figure 2. IP Media (IPML) Class Diagram



5.2.2 Channel Class

The Channel class' main role 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 7. Refer to the source code for method information.

Table 7. 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

Table 7. Channel Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_pIPP	public	IPProtocol*	Proprietary protocol object to control the calls with the remote gateway
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 Configuration class' main role 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 8. Refer to the source code for method information.

Table 8. Configuration Class Attributes

Name	Access Privilege	Type	Description
m_pstnProtocol	public	char	PSTN protocol to be used, read from the configuration file and updated during initialization. It is not changed until demo termination. It is accessed while opening the PSTN Board and Digital PSTN devices.
m_remoteHostPortInfo	public	IPM_PORT_INFO	Structure that contains the remote host IP address and port, read from the configuration file and updated during initialization. It is not changed until demo termination. It is accessed while sending IP messages to the remote gateway.
m_localHostPortInfo	public	IPM_PORT_INFO	Structure that contains the local host port, read from the configuration file and updated during initialization. It is not changed until demo termination. It is accessed while initializing the server.

Table 8. Configuration Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_logFArr	public	char	Log level that the logs will be printed with, read whenever something is printed to the log. It is updated by command line option during initialization or by the user request when pressing the d or D key during run time. The log level is the same for all log objects.
m_firstCallId	public	long	The lowest CallId (IP call control identifier) number allowed, read from the configuration file during initialization. It is not changed until demo termination. The Call IDs range must be different at the two gateways.
m_lastCallId	public	long	The highest CallId (IP call control identifier) number allowed.
m_QoSFlag	public	int	If the QoS feature is enabled by the -q command line option, it is read from the configuration file during initialization and not changed until demo termination. It is accessed before calling any QoS function to check if that function should be called.
m_stage	private	unsigned char	The stage of parsing the configuration file
m_line	private	int	The line in the configuration file currently being parsed
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)
m_userChannels	public	unsigned int	Indicates the number of channels that the demo will work with. Updated after reading the -n command line option and remains unchanged until demo termination.
m_DigitMaps	private	int [MAX_DIGIT_MAPS]	Array of “digit maps” to collect DTMFs in order to send them over the IP. The digit map includes only number of digits for each loop.

Table 8. Configuration Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_DigitMapNum	private	int	Current digit map read
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. It is updated while getting data from the configuration file.

5.2.4 DigitalPSTNBoard Class

The DigitalPSTNBoard class' main role 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 9. Refer to the source code for method information.

Table 9. 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 DigitalPSTNDevice class' main role 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 10. Refer to the source code for method information.

Table 10. DigitalPSTNDevice Class Attributes

Name	Access Privilege	Type	Description
m_lineDevice	public	LINEDEV	Line device of the PSTN device - valid after opening the PSTN device

Table 10. DigitalPSTNDevice Class Attributes (Continued)

Name	Access Privilege	Type	Description
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 GWCall class' main role is to control all resources related to a call. It contains all the resources needed to establish a call:

- PSTNCallControl
- IPMediaDevice
- IPCallControl

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

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

Table 11. 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_pIPCC	public	IPCallControl*	IP call control object of the call, handling the IP proprietary call control
m_pLog	public	Log*	Log object of the call
m_currentState	private	E_StateMachine	Current channel state
m_LastDigit	private	int	Points to the last index reached in m_DTMFBuffer
m_DTMFBuffer	private	char[MAX_DIGITS]	Used to store DTMFs to send over the IP. This DTMF "digit map" is used to avoid sending the DTMFs received from the PSTN one by one, which leads to many unnecessary messages sent over the IP. The received DTMFs are stored according to a digit map defined in the CFG file and sent only when the expected digit count is reached. The digit map only defines how many digits should be stored before being sent.

Table 11. GWCall Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_NumOfDigits	private	int	Number of digits expected from the PSTN this loop
m_DigitMapNum	private	int	Current digit map to be matched to the digits from the PSTN
m_ChangeCodecFlag	private	int	Binary flag. First bit is “1” if, and only if, IPMEDIA_STOP arrived when changing the codec while in a call. The second bit is set to “1” when the remote side sends IPEV_CODEC_CHANGED.
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 but is the same, but for the IP side.

5.2.7 IPCallControl Class

The IPCallControl class’ main role is to provide the IP proprietary protocol functionality interface. It controls one IP call on a specific IPProtocol and reflects the call status to the other classes.

The IPCallControl class attributes are described in Table 12. Refer to the source code for method information.

Table 12. IPCallControl Class Attributes

Name	Access Privilege	Type	Description
m_CallId	private	long	Call ID of this object
m_pIpp	private	IPProtocol*	IPProtocol object containing this object
m_State	private	E_IPState	The current state
m_pLog	private	Log*	Log object for printing

5.2.8 IPMediaBoard Class

The IPMediaBoard class’ main role is to initiate the IP Media board. It manages the IP Media device database and contains all the IP Media devices available in the system.

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

Table 13. 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 to a Channel.

5.2.9 IPMediaDevice Class

The IPMediaDevice class' main role is to provide IPML functionality for 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 14. Refer to the source code for method information.

Table 14. IPMediaDevice Class Attributes

Name	Access Privilege	Type	Description
m_currentState	private	E_StateMachine	The current state of the channel
m_mediaInfo	private	IPM_MEDIA_INFO (see *ipmlib.h*)	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_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.
m_QoSAlarmFile	public	FILE*	Pointer to the QoS statistics file
m_WoSAlarmFileName	public	Char*	The QoS statistics file name - used when opening the file
m_LocalVoiceMediaInfo	private	IPM_MEDIA_INFO	The local media information for voice - updated after getting the local media information (using the IPML API).
m_RemoteVoiceMediaInfo	private	IPM_MEDIA_INFO	The voice media information of the remote gateway - updated when getting an offering or answering from the remote gateway.
m_LocalFaxMediaInfo	private	IPM_MEDIA_INFO	The local media information for fax
m_RemoteFaxMediaInfo	private	IPM_MEDIA_INFO	The fax media information of the remote gateway

Table 14. IPMediaDevice Class Attributes (Continued)

Name	Access Privilege	Type	Description
m_DTMEMode	private	E_DTMFMode	The mode that should be used for sending DTMFs - inband, out of band, or RFC2833
m_FaxMode	private	E_FaxMode	The fax codec: T38, G711, or no fax (voice codec)
m_FaxTone	private	IPM_FAX_SIGNAL	The detected fax event
m_isFree	public	bool	Indicates if the IPMedia device is free.

5.2.10 IPMsg Class

The IPMsg class is part of the proprietary call control. It describes a message that can be sent over the IP and provides an interface to prepare any message needed by the demo to be sent over IP.

The IPMsg class attributes are described in Table 15. Refer to the source code for method information.

Table 15. IPMsg Class Attributes

Name	Access Privilege	Type	Description
m_MsgType	private	E_IPMsg	Type of message to be sent over the IP
m_CallId	private	long	CallID this message is connected to
m_DigitInfo	private	IPM_DIGIT_INFO	Digit information for sending DTMF over IP
m_MediaInfo		IPM_MEDIA_INFO	RTP, RTCP, and codec information
m_MediaMode	private	E_MediaMode	Voice or data
m_CodecInfo	private	IPM_CODER_INFO	Codec information
m_DropReason	private	E_DropReason	Reason for dropping the call

5.2.11 IPProtocol Class

The IPProtocol class' main role is to provide the IP proprietary protocol interface. It implements the IP proprietary protocol unit and manages all calls related to this class.

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

Table 16. IPProtocol Class Attributes

Name	Access Privilege	Type	Description
m_Handle	private	unsigned int	The Call ID of this object
m_IPCallControl	private	IPCallControlMap	Map connecting IPCallControl to a CallID
m_State	private	static IPProtocolMgr*	Manager object of all IPProtocol objects
m_pLog	private	Log*	A Log object for printing

5.2.12 IPProtocolMgr Class

The IPProtocolMgr class' main role is to:

- initiate and manage the IPCallControl objects
- initialize the client and server to communicate between gateways
- generate the call ID

The IPProtocolMgr class contains all the IPProtocol instances and reflects the connection status to the other classes.

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

Table 17. IPProtocolMgr Class Attributes

Name	Access Privilege	Type	Description
m_IPProtocols	private	IPProtocolQueue	List of all IPProtocol objects in the system
m_IppToCallId	private	IPProtocolMap	Binds an IPProtocol object to a CallId number
m_CallIdPool	private	char [CALL_ID_POOL_SIZE]	Marks which CallId numbers are available
m_FirstCallId	private	long	The lowest CallId number allowed
m_LastCallId	private	long	The highest CallId number allowed
m_pLog	private	Log*	A Log object for printing

5.2.13 PSTNCallControl Class

The PSTNCallControl class' main role is to provide a Global Call functionality interface to manage a call. It reflects the call status to the other classes.

The PSTNCallControl class attributes are described in Table 18. Refer to the source code for method information.

Table 18. 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_localPhoneNumber	public	char	Local Phone number to use with **gc_MakeCall( )**
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

5.2.14 R4Device Class

The R4Device class' main role is to provide all common functionality for all R4 devices. It is a base class that contains all the common attributes for all R4 devices.

The R4Device class attributes are described in Table 19. Refer to the source code for method information.

Table 19. 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

5.2.15 R4LogicalBoard Class

The R4LogicalBoard class' main role 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 20. Refer to the source code for method information.

Table 20. 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_boardDevice	protected	LINEDEV	Returned when opening the board by **gc_Open( )** and used to get the devices found on it by calling the function **ATDV_SUBDEVS( )** and to close the board.
m_boardHandleS	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

5.2.16 ResourceManager Class

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

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

The ResourceManager class attributes are described in Table 21. Refer to the source code for method information.

Table 21. 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_pIPProtocolMgr	public	static IPProtocolMgr*	Manages the IPProtocol instances. It allocates them, creates call IDs for them, initializes the transport module: client and server for send and receive messages over IP. This instance is created at the demo initialization and is accessed whenever the systems needs to allocate a new IPProtocol or call id object, or to map a call ID to an IPProtocol. There is one instance of this object in the system.
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.

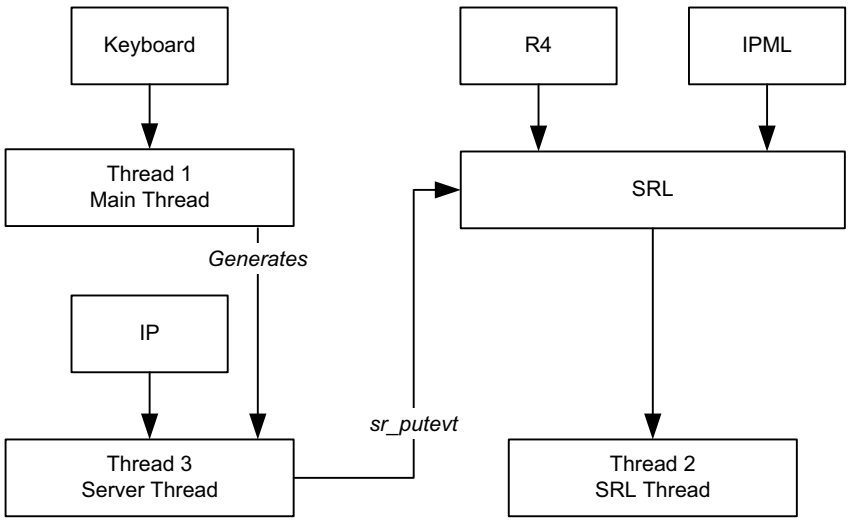
Table 21. ResourceManager Class Attributes (Continued)

Name	Access Privilege	Type	Description
IPPHandleToChannel	public	static unsigned int	Maps IPProtocol handle (this handle is actually the IP call ID) to channel.
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_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.

5.3 Threads

The IP Media (IPML) demo operates with three threads, as shown in Figure 3.

Figure 3. IP Media (IPML) Demo Threads



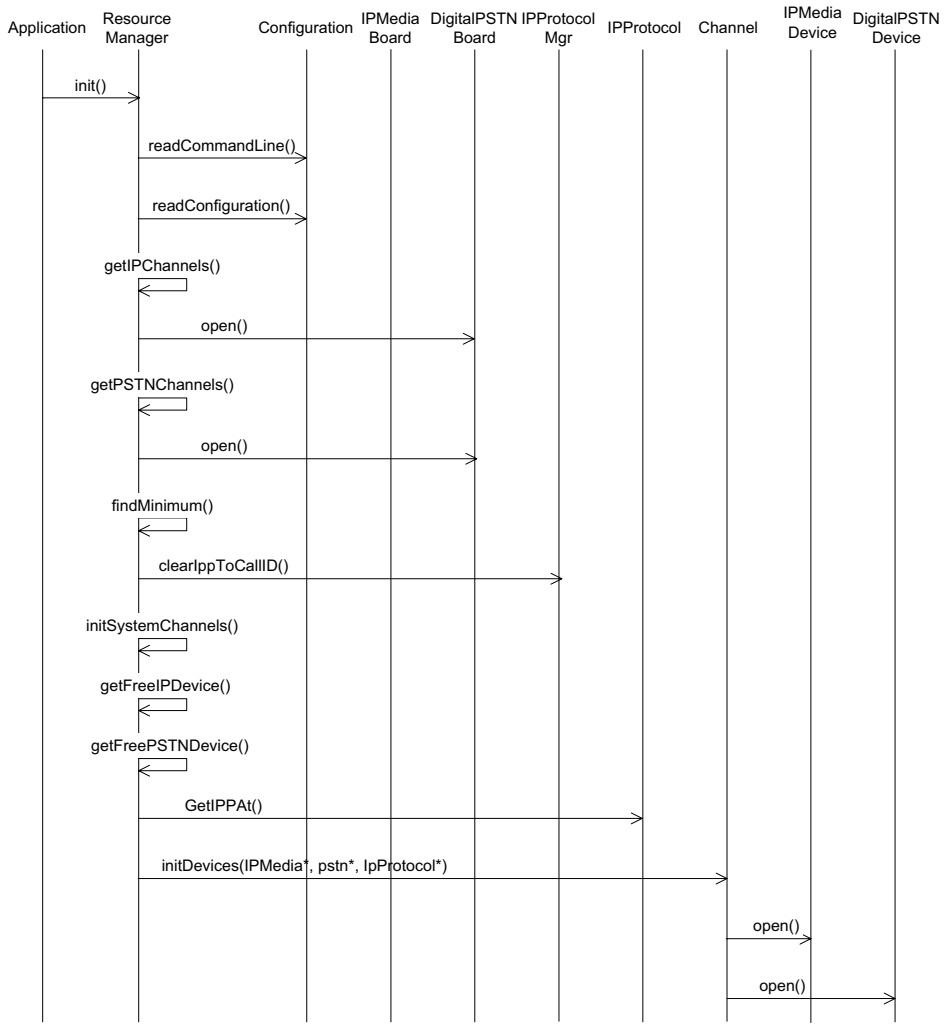
The threads are created as follows:

- The first (main) thread 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.
- The third thread is the server thread that waits for messages from the IP and puts events on the SRL queue. This thread is created at the initialization of the IP Protocol Manager and closed at its destruction.

5.4 Initialization

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

Figure 4. IP Media (IPML) System Initialization



The Resource Manager calls the **init( )** function, which does the following:

1. Reads the command line options
2. Reads and parse the configuration file and prints the configuration
3. Gets the resources available in the system:
   a. Gets the number of IP channels in the system
   b. Gets the number of PSTN channels in the system
   c. Finds the minimum between the system channels and the user request
4. Initializes the IP Protocol Manager
5. Looks for a free Ipmedia device and returns a pointer to it
6. Opens the Ipmedia device and if the open succeeds returns a pointer to it

7. Looks for a free PSTN device and returns a pointer to it
8. Opens the PSTN device and if the open success returns a pointer to it
9. Looks for a free IP Protocol (IPP) device and returns a pointer to it
10. Opens the IPP device and if the open success returns a pointer to it
11. Initializes the devices on the channel

5.5 Event Handling

This section describes how the IP Media (IPML) demo handles events. It contains the following topics:

5.5.1 Event Mechanism

The IP Media (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( )**.

Refer to Chapter 6, "Demo State Machines" for more detailed event handling information.

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. Gets the event device handle, by calling **PDLsr_getevtdev( )**
2. Gets the channel number related to the event, from the global array (HandleToChannel[ ])
3. Updates the METAEVENT structure by calling **gc_GetMetaEvent( )**

4. Gets the event type, by calling **PDLsr_getevttype( )**

5.5.4 Handling Application Exit Events

Normal application exit events don't enter the SRL. The **main( )** function calls **PDLSetApplicationExitPath( )** before initialization. In Linux, this function sets the signals (SIGINT, SIGTERM, SIGABRT) for making the appropriate exit from the application. In Windows, this function enables the detection of CTRL_CLOSE_EVENT (closing the window).

Demo State Machines 6

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

6.1 GWCall State Machine - Inbound Call from IP

This section describes the state machine for an inbound call from the IP. It contains the following topics:

6.1.1 GWCall State Machine Description - Inbound from IP

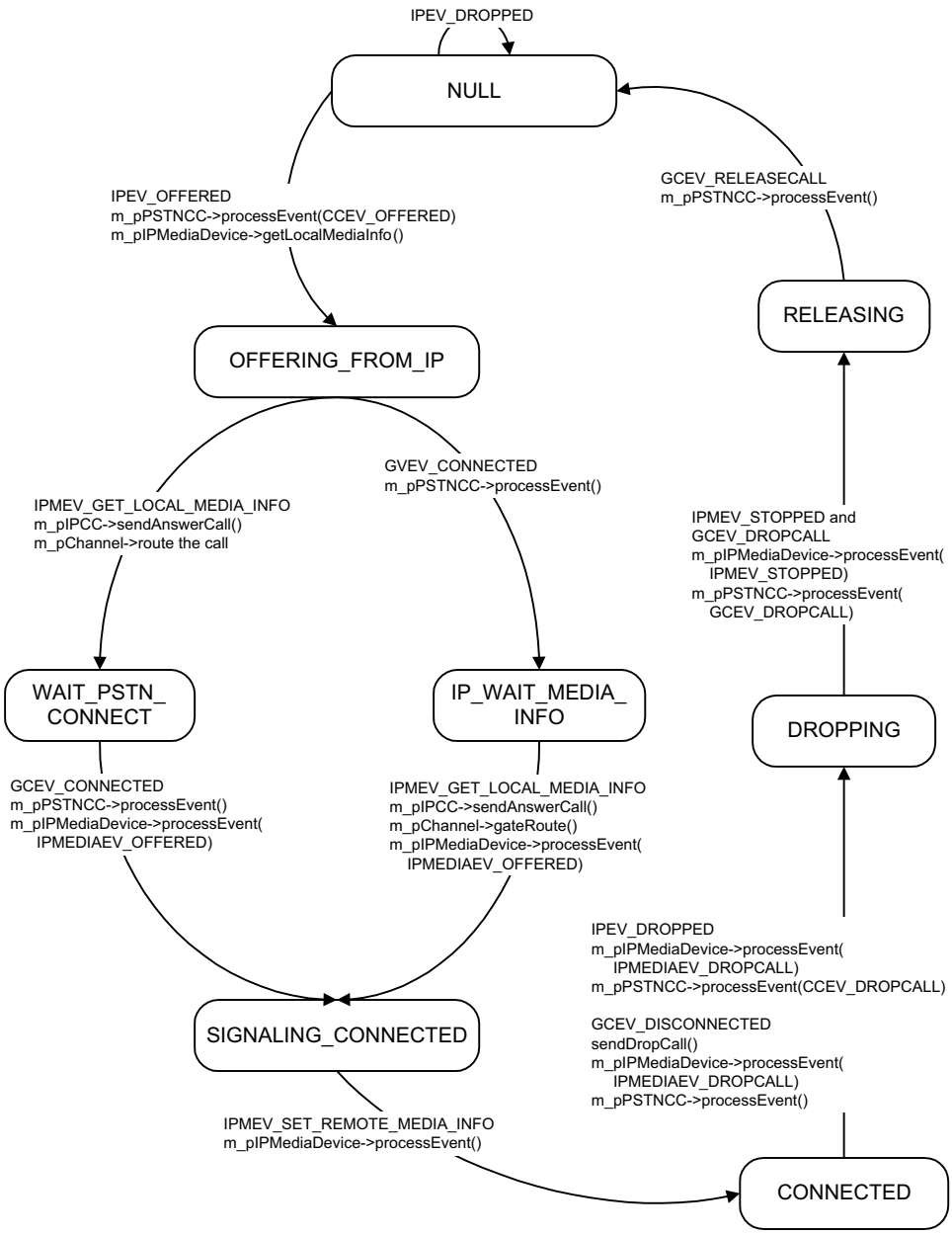
All channels are initialized to the NULL state upon application start.

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 for an inbound call from the IP.

Figure 5. GWCall State Machine - Inbound Call from IP



6.1.2 GWCall::gateNull State

The application waits in the gateNull state for an IPEV_OFFERED event (from IPCallControl). Upon receipt of the event, the application calls **processEvent(CCEV_OFFERED)** from the PSTNCallControl module. See Section 6.5, “PSTNCallControl State Machine”, on page 61 for a description of the PSTNCallControl state machine. The application then calls the function

getLocalMediaInfo( ) from the IPMediaDevice module. See Section 6.4, “IPMediaDevice State Machine”, on page 59 for a description of the IPMediaDevice state machine.

The state transitions to gateOfferingFromIP.

The application may also receive a GCEV_DETECTED or GCEV_OFFERED event. See Section 6.2, “GW Call State Machine - Inbound Call from PSTN”, on page 52 for a description of how the application deals with these events.

If either the PSTNCallControl module or the IPMediaDevice module return NULL, the application calls the function **sendDropCall(BUSY)**, which sends a drop call message to the remote side through the IP call control stack.

6.1.3 GWCall::gateOfferingFromIP State

The application waits for either a GCEV_CONNECTED or an IPMEV_GET_LOCAL_MEDIA_INFO event.

If it receives a GCEV_CONNECTED event, the application calls **processEvent(GCEV_CONNECTED)** from the PSTNCallControl module. The call state transitions to gateIPWaitMediaInfo and the application routes the call.

If the application receives an IPMEV_SET_LOCAL_MEDIA_INFO event, it calls **sendAnswerCall( )** from the IPCallControl module. The call state transitions to gateWaitPSTNConnect and the application routes the call.

If the application receives an IPEV_DROPPED or a GC_TASKFAIL event, it calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module, and the state transitions to gateDropping.

6.1.4 GWCall::gateIPWaitMediaInfo State

The application waits for an IPMEV_GET_LOCAL_MEDIA_INFO event. Upon receipt of the event, it gets the local media information and calls **sendAnswerCall( )** from the IPCallControl module. The application then calls **gateRoute( )** from the Channel module and calls **processEvent(IPMEDIAEV_OFFERED)** from the IPMediaDevice module. The state transitions to gateSignalingConnected.

If the application receives an IPEV_DROPPED event, it calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

6.1.5 GWCall::gateWaitPSTNConnect State

The application waits for a GCEV_CONNECTED event. Upon receipt of the event, it calls **processEvent(GCEV_CONNECTED)** from the PSTNCallControl module to and calls

processEvent(IPMEDIAEV_OFFERED) from the IPMediaDevice module. The call state transitions to gateSignalingConnected.

If the application receives an IPEV_DROPPED event, it calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

6.1.6 GWCall::gateSignalingConnected State

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

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

If the application receives an IPEV_DROPPED event, it calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

6.1.7 GWCall::gateConnected State

The application waits for a IPEV_DROPPED or a GCEV_DISCONNECTED event. In the case of IPEV_DROPPED, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module, and calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

In the case of GCEV_DISCONNECTED, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and calls **processEvent(GCEV_DISCONNECTED)** from the PSTNCallControl. The call state transitions to gateDropping.

The application may also receive an event requesting that it switch from voice to fax. Refer to Section 6.3, “GWCall State Machine - Switching Between Voice/Fax”, on page 57 for more detailed information.

6.2 GW Call State Machine - Inbound Call from PSTN

This section describes the state machine for an inbound call from the PSTN. It contains the following topics:

- GWCall::gateDetectedFromPSTN State
- GWCall::gateOfferingFromPSTN State
- GWCall::gatePSTNWaitMediaInfo State
- GWCall::gateWaitPSTNAnswer State
- GWCall::gatePSTNConnected State
- GWCall::gateSignalingConnected State
- GWCall::gateConnected State

6.2.1 GWCall State Machine Description - Inbound from PSTN

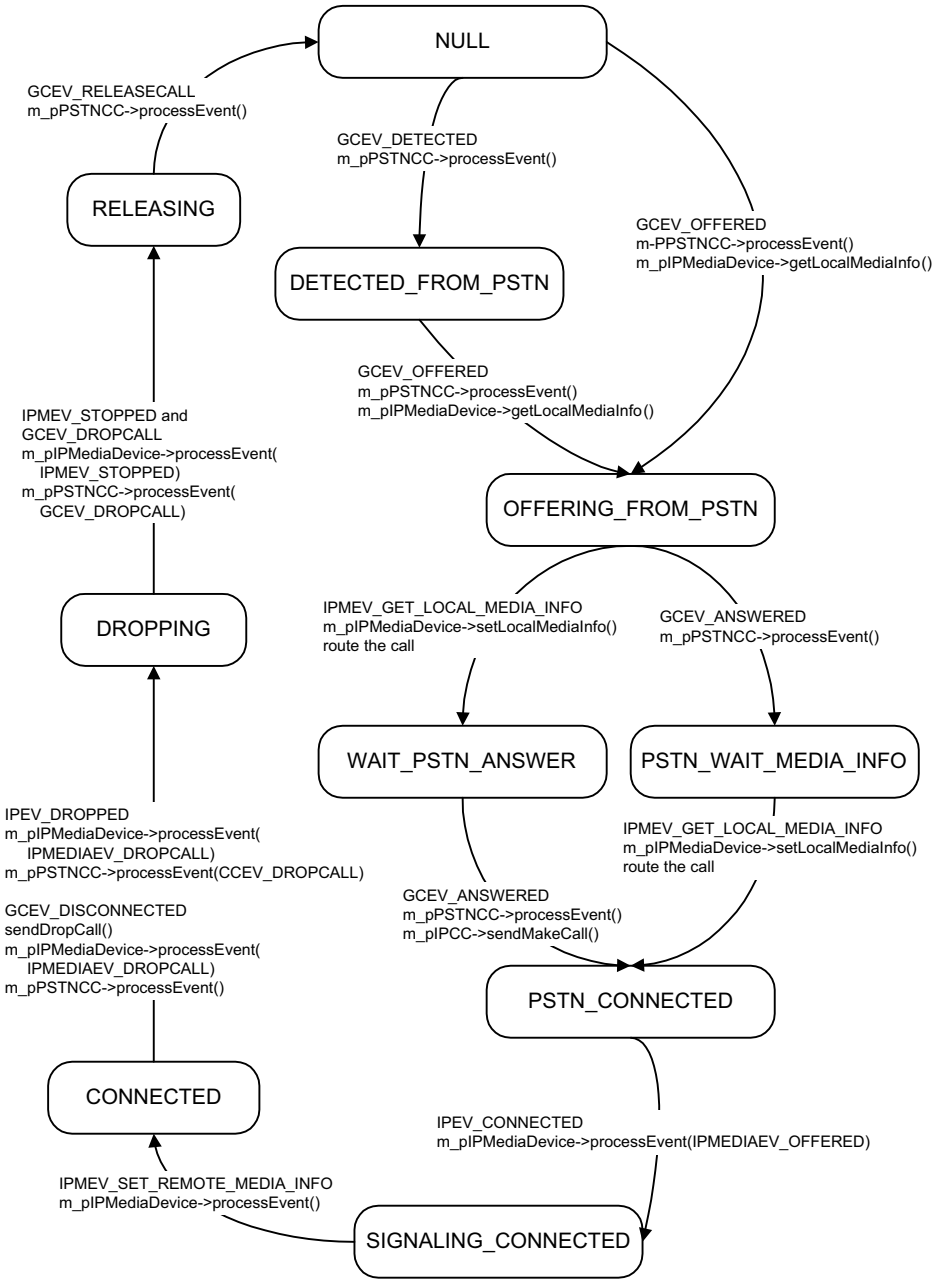
All channels are initialized to the NULL state upon application start.

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 for an inbound call from the PSTN.

Figure 6. GWCall State Machine - Inbound Call from PSTN



6.2.2 GWCall::gateNull State

The application waits for a GCEV_OFFERED or GCEV_DETECTED event. In the case of GCEV_DETECTED, the application calls **processEvent(GCEV_DETECTED)** from the

PSTNCallControl module and the call state transitions to gateDetectedFromPSTN. See Section 6.5, "PSTNCallControl State Machine", on page 61 for a description of the PSTNCallControl state machine.

In the case of GCEV_OFFERED, the application calls **processEvent(GCEV_OFFERED)** from the PSTNCallControl module and calls **getLocalMediaInfo( )** from the IPMediaDevice to get the local media information. The call state transitions to gateOfferingFromPSTN.

The application may also receive an IPEV_OFFERED event. See Section 6.1, "GWCall State Machine - Inbound Call from IP", on page 49 for a description of how the application deals with these events.

If either the PSTNCallControl module or the IPMediaDevice module return NULL, the application calls the function **sendDropCall(BUSY)**, which sends a dropCall message to the remote side through the IP call control stack.

6.2.3 GWCall::gateDetectedFromPSTN State

The application waits for a GCEV_OFFERED event. It calls **processEvent(GCEV_OFFERED)** from the PSTNCallControl module and calls **getLocalMediaInfo( )** from the IPMediaDevice module. The call state transitions to gateOfferingFromPSTN.

6.2.4 GWCall::gateOfferingFromPSTN State

The application waits for either a GCEV_ANSWERED or IPMEV_GET_LOCAL_MEDIA_INFO event. In the case it receives a GCEV_ANSWERED event, the application calls **processEvent(GCEV_ANSWERED)** from the PSTNCallControl module. The call state transitions to gatePSTNWaitMediaInfo.

In the case it receives IPMEV_GET_LOCAL_MEDIA_INFO, the application tells the IPMediaDevice module to get the local media information and update the media device, by calling **setLocalMediaInfo( )**. The call state transitions to gateWaitPSTNAnswer, and the call is routed.

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

6.2.5 GWCall::gatePSTNWaitMediaInfo State

The application waits for an IPMEV_GET_LOCAL_MEDIA_INFO event. Upon receipt of the event, the application calls **setLocalMediaInfo( )** from the IPMediaDevice module and then calls **sendMakeCall( )** from the IPCallControl module. The call state transitions to gatePSTNConnected, and the call is routed.

If the application receives either GCEV_DISCONNECTED or IPMEV_ERROR, it calls **processEvent(GCEV_DISCONNECTED)** from the PSTNCallControl module and the call state transitions to gateDropping.

If the application receives an IPEV_DROPPED event, it calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module and the call state transitions to gateDropping.

6.2.6 GWCall::gateWaitPSTNAnswer State

The application waits for a GCEV_ANSWERED event. Upon receipt of the event, the application calls **processEvent(GCEV_ANSWERED)** from the PSTNCallControl module. The application sets the information needed for the makeCall message and calls **sendMakeCall( )** from the IPCallControl module. The call state transitions to gatePSTNConnected.

6.2.7 GWCall::gatePSTNConnected State

The application waits for an IPEV_CONNECTED event. Upon receipt of the event, the application calls **processEvent(IPMEDIAEV_OFFERED)** from the IPMediaDevice module. The call state transitions to gateSignalingConnected.

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

If the application receives an IPEV_DROPPED event, it calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module and the call state transitions to gateDropping.

6.2.8 GWCall::gateSignalingConnected State

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

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

If the application receives an IPEV_DROPPED event, it calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

6.2.9 GWCall::gateConnected State

The application waits for a IPEV_DROPPED or a GCEV_DISCONNECTED event. In the case of IPEV_DROPPED, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module, and calls **processEvent(CCEV_DROPCALL)** from the PSTNCallControl module. The call state transitions to gateDropping.

In the case of GCEV_DISCONNECTED, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice module and **processEvent(GCEV_DISCONNECTED)** from the PSTNCallControl. The call state transitions to gateDropping.

The application may also receive an event requesting that it switch from voice to fax. Refer to Section 6.3, "GWCall State Machine - Switching Between Voice/Fax", on page 57 for more detailed information.

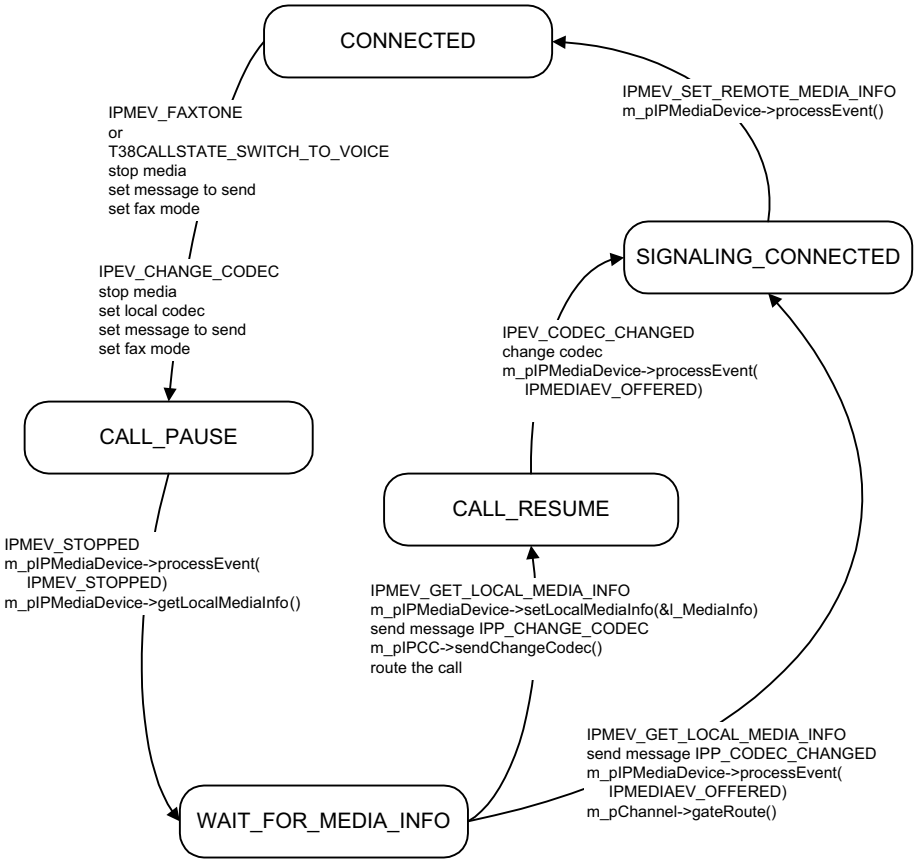
6.3 GWCall State Machine - Switching Between Voice/Fax

This section describes the state machine for switching between voice and fax during a call. It contains the following topics:

6.3.1 GWCall State Machine Description - Voice/Fax

The following state diagram describes the call states for the GWCall class for switching between voice and fax.

Figure 7. GWCall State Machine - Switching Between Voice/Fax



6.3.2 GWCall::gateConnected State

The application waits for a fax event or a change codec event. If it receives IPMEV_FAXTONE (in voice mode) or T38CALLSTATE_SWITCH_TO_VOICE (in fax mode), it stops the media, sets the message to send to the remote gateway via the IPProtocol, and sets the media mode. If it receives IPEV_CHANGE_CODEC, it stops the media, sets the local codec, sets the message to send to the remote gateway via the IPProtocol, and sets the media mode.

In the case of IPMEV_FAXTONE, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice to stop the media, and then calls **getDefaultFaxCodec( )** to get the fax codec as defined in the *ipmedia_r4.cfg* file. It calls **setFaxCodec(true)** to pass the CED tone to the remote gateway. The application sends an IPP_CHANGE_CODEC message to the remote gateway. The call state transitions to gateCallPause.

In the case of IPMEV_T38CALLSTATE_SWITCH_TO_VOICE, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice to stop the media, and then to calls **setFaxMode(FAX_NONE)**, **getDefaultVoiceCodec( )** to get the voice codec as defined in

the *ipmedia_r4.cfg* file, and **setVoiceCodec( )**. The application sends an IPP_CHANGE_CODEC message to the remote gateway. The call state transitions to gateCallPause.

In the case of IPEV_CHANGE_CODEC, the application calls **processEvent(IPMEDIAEV_DROPCALL)** from the IPMediaDevice to stop the media, and then retrieves the codec and remote media information from the message. If the media mode is fax, the application gets the fax mode (G.711 over UDP or T.38 over RTP). If the fax mode is T.38, the application calls **setRemoteFaxMediaInfo( )** from the IPMediaDevice. If the fax mode is G.711, the application calls **setRemoteVoiceMediaInfo( )** from the IPMediaDevice. If the media mode is voice, the application calls **setFaxMode(FAX_NONE)**, **setVoiceCodec( )**, and **setRemoteVoiceMediaInfo( )**. The application sends an IPP_CODEC_CHANGED message to the remote gateway. The call state transitions to gateCallPause.

6.3.3 GWCall::gateCallPause State

The application waits for an IPMEV_STOPPED event. Upon receipt of the event, the application calls **processEvent(IPMEV_STOPPED)** from the IPMediaDevice and then calls **getLocalMediaInfo( )**. The call state transitions to gateWaitForMediaInfo.

6.3.4 GWCall::gateWaitForMediaInfo State

The application waits for an IPMEV_GET_LOCAL_MEDIA_INFO event. Upon receipt of the event, the application calls **setLocalMediaInfo( )** from the IPMediaDevice.

6.3.5 GWCall::gateResume State

The application waits for an IPEV_CODEC_CHANGED event. Upon receipt of the event, the application updates the remote codec according to the media mode (fax or voice). If the media mode is fax, the application calls **setRemoteFaxMediaInfo( )** from the IPMediaDevice. If the media mode is voice, the application calls **setRemoteVoiceMediaInfo( )** from the IPMediaDevice. It then calls **processEvent(IPMEDIAEV_OFFERED)** from the IPMediaDevice and the call state transitions to gateSignalingConnected.

6.4 IPMediaDevice State Machine

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

6.4.1 IPMediaDevice State Machine Description

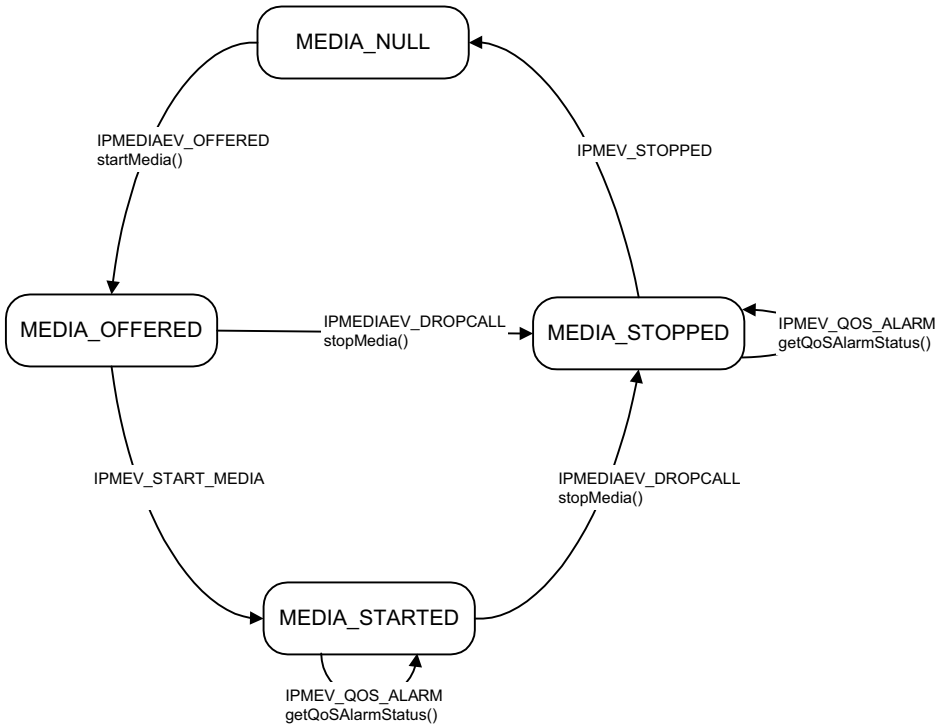
All channels are initialized to the NULL state upon application start.

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 IPMediaDevice class.

Figure 8. IPMediaDevice State Machine



6.4.2 IPMediaDevice::mediaNull State

The application waits for an IPMEDIAV_OFFERED event. Upon receipt of the event, it calls **startMedia( )**, which gets the media information from the local and remote media structures and calls **ipm_StartMedia( )** using the information from these structures. The state transitions to mediaOffered.

6.4.3 IPMediaDevice::mediaOffered State

The application waits for an IPMEV_START_MEDIA event. Upon receipt of the event, the call state transitions to mediaStarted.

If the application should receive an IPMEDIAEV_DROPCALL event, it calls **stopMedia( ),** which unroutes the call and calls **ipm_Stop( )**. The state transitions to mediaStopped.

6.4.4 IPMediaDevice::mediaStarted State

The application waits for an IPMEDIAEV_DROPCALL event. Upon receipt of the event, if Quality of Service was enabled the application calls **getSessionInfo( )** to gather the Quality of Service statistics. The application calls **stopMedia( )** and the state transitions to mediaStopped.

The application can also receive an IPMEV_QOS_ALARM event. It calls **getQoSAlarmStatus( )** to get the alarm type. The state remains unchanged.

6.4.5 IPMediaDevice::mediaStopped State

The application waits for an IPMEV_STOPPED event. Upon receipt of the event, the state transitions to mediaNull. If the application receives an IPMEDIAEV_DROPCALL event, it does nothing and continues to wait in the current state.

The application can also receive an IPMEV_QOS_ALARM event. It calls **getQoSAlarmStatus( )** to get the alarm type. The state remains unchanged.

6.5 PSTNCallControl State Machine

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

6.5.1 PSTNCallControl State Machine Description

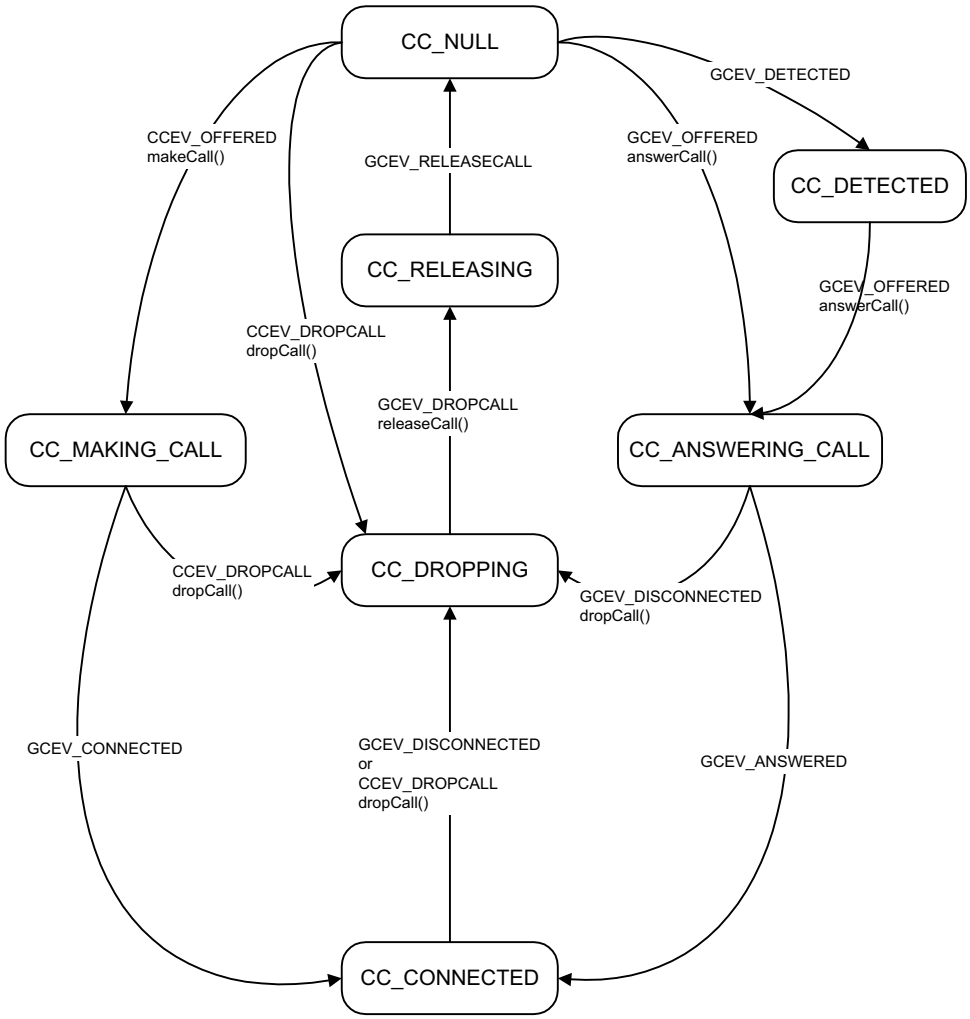
All channels are initialized to the NULL state upon application start.

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 PSTNCallControl class.

Figure 9. PSTNCallControl State Machine



6.5.2 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.

6.5.3 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.

6.5.4 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.

6.5.5 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.

6.5.6 PSTNCallControl::CCConnected State

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

6.5.7 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.

6.5.8 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

Asynchronous mode: An operating mode of certain DM3 kernel or host library function calls. Asynchronous mode is a non-blocking mode that is typically used when a function involves operation on a remote processor (e.g., a host library function that initiates a kernel operation on a DM3 platform's Control Processor) or when data movement via the MMA and global memory is required. In asynchronous mode a value signifying "pending" status is immediately returned to the calling function, and the actual completion or failure of the operation is reported to the caller via a DM3 result message.

Codec: see COder/DECoder

COder/DECoder: A circuit used on Dialogic boards 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.

Control Processor (CP): A processor that is present on the motherboard of every DM3 platform for the management of the SCbus/CT bus, for host communication configuration, and to implement some of the features of DM3 Resources. All current DM3 platforms use an Intel i960 as the Control Processor.

DM3 kernel: Firmware that is present on each processor on a DM3 platform to support configuration management, host communication, inter processor communication and control, and SCSA firmware services among others. There may be more than one version of the DM3 kernel firmware depending on the type of processor (e.g. CP vs. DSP or RISC signal processor) and the intended application of each processor.

DM3 load module: A loadable block of executable firmware for a particular processor on a DM3 platform. A DM3 Resource may be packaged as one or more DM3 load modules.

DTMF: See 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. In

the context of DSP programming (e.g. DM3 component development), the period defined by the sample rate of the signal data.

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: An H.323 Gateway (GW) is an endpoint on the Internet which provides for real-time, two-way communications between H.323 Terminals on the Network and other ITU Terminals on a wide area network, or to IP Telephony clients.

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 station to communicate over network.

Parameter: A type of datum that is passed to or from a component via a DM3 message. Parameters generally have two elements, a type and a specific value. Parameters passed to a component affects the operational characteristics of the component, and parameters passed from a component can be used to report the operating state of the component. As an example, the standard Player component accepts a parameter called ParmDuration, which specifies the length of the file that is being played back, and can report its current state (e.g. playing, paused, etc.) via a parameter called ParmState.

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.

Receiving Gateway: accepts a connection from an Emitting Gateway. It calls a G3FE and provides IFT service to the called G3FE.

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

SCbus: The standard bus for communication within a SCSA node. The architecture of SCbus includes a 16-wire TDM data bus that operates at 2, 4 or 8 Mbps and a serial message bus for control and signaling. DM3 platforms provide an SCbus interface for interconnection of multiple DM3 platforms, or connection to other SCSA-compatible hardware. The DM3 platform supports timeslot bundling for high bandwidth, and can access up to 256 of the 2048 SCbus timeslots via two SC4000 ASICs.

Signal Processor (SP): An embedded processor on a DM3 platform that is used to execute signal processing algorithms. The DM3 architecture supports multiple types of SPs, including RISC processors as well as general purpose DSPs, and platforms may be configured with a mixed complement of SP types.

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.

Synchronous mode: An operating mode of certain DM3 kernel or host library function calls. Synchronous mode is a blocking mode that is typically used only when a function executes on the local processor using data that is also local to the processor (i.e. that does not require any data movement via the MMA).

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

Symbols

A

C

D

E

F

G

T

U

W