intel®

IP Media Server (Global Call) Demo Guide

for Windows Operating Systems

November 2003

intel®

intel. ®

Contents

intel®

Figures

intel®

Tables

int_el ®

Revision History

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

Document No.	Publication Date	Description of Revisions
05-2065-002	November 2003	Fax not supported in SR 6.0 PCI Windows demo - removed all references
05-2065-001	September 2003	Initial version of document.

intel.

About This Publication

This section describes the purpose of the guide, the intended audience, and provides references to other documents that may be useful to the user.

- Purpose
- Intended Audience
- How to Use This Publication
- Related Information

Purpose

This guide provides information on the IP Media Server (Global Call) demo that is available with your Intel® Dialogic® System Release 6.0 on PCI for Windows. The 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 a media server application using the Global Call API. Developers should be familiar with the C++ programming language and the Windows* programming environment.

This information is intended for:

- Distributors
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

How to Use This Publication

Refer to this publication after you have installed the hardware and the system software.

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

The information in this guide is organized as follows:

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

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

Related Information

See the following for more information:

- *Intel® Dialogic® System Release 6.0 on PCI for Windows Release Guide*
- *Intel Dialogic and Intel NetStructure for DM3 Architecture for PCI on Windows Configuration Guide*
- *Global Call IP Technology Guide*
- *Voice Software Reference: Programmer's Guide for Windows*
- *Voice Software Reference: Standard Runtime Library for Windows*
- *http://developer.intel.com/design/telecom/support/* (for technical support)
- *http://www.intel.com/network/csp/* (for product information)

intel®

Demo Description 1

This chapter describes the basic features of the IP Media Server (Global Call) demo.

The IP Media Server (Global Call) demo is an object-oriented host-based application that demonstrates using the Global Call API to build an IP media server, providing voice and fax services via IP technology. 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 Media Server (Global Call) demo supports the following features:

- Voice service
- CSP barge in
- Configuration file
- Command line options

Note: The IP Media Server (Global Call) demo does not function as a gateway. Therefore, it can only answer calls from the IP network. Gateway functionality can be added by writing additional software code within the IP module that will allow it to make outgoing calls to the IP network, and connecting a gateway to interface with the PSTN.

The IP Media Server (Global Call) demo is a cross-OS demo, designed to eventually run under both 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 about the PDL refer to the source code in the *pdl_win* directory.

intel

System Requirements 2

This chapter discusses the system requirements for running the IP Media Server (Global Call). It contains the following topics:

2.1 Hardware Requirements

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

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

intel.

Preparing to Run the Demo   **3**

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

3.1 Editing Configuration Files

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

- File Location
- Editing the IPMediaServer.cfg Configuration File

3.1.1 File Location

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

C:\Program Files\Dialogic\Demos\IPDemos\IPMediaServer

3.1.2 Editing the IPMediaServer.cfg Configuration File

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

ipProtocolName
> The IP protocol for opening IP line devices. Values are: H323, SIP, both.

DTMFmode
> Specifies how DTMF tones are transmitted. Possible values are: OutofBand (usually used with low bandwith coders, e.g., GSM), InBand (usually used with G.711 coders), RFC2833.
>> *Note:*   OutofBand is used for H.323 only.

Capability
> Describes the transmit and receive coders. See the *Global Call IP Technology Guide* for specific information about coder support. The parameters are as follows:
> - TxType – The transmit voice coder
> - TxFramesPerPkt – Specify the number of frames per packet for the selected Tx coder
> - TxVAD – Specify if VAD is active for the selected Tx coder
> - RxType – The receive voice coder
> - RxFramesPerPkt – Specify the number of frames per packet for the selected Rx coder
> - RxVAD – Specify if VAD is active for the selected Rx coder

Note: The G.711 coder defines frames per packet using the packet size in milliseconds, i.e. 10, 20, or 30 milliseconds. Refer to Sample Configuration File for the correct syntax for all 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 – monitors the number of lost IP packets during a call
> - MediaAlarmJitter – monitors the jitter (as defined in RFC 1889) during IP transmission

QoS Attributes

> The threshold for each QoS parameter is measured with the following 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 – used to detect failure condition, together with DebounceOn (expressed as a percentage of failures)
> - Percent_Success – used to detect failure recovery, together with DebounceOff (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

Sample Configuration File

```
##########################################################################################
# IP Protocol :
#    The IP Protocol used for  opening the IP Line devices, values: H323, SIP, both
#
# DTMFmode
#possible options:
#OutOfBand, inband, rfc2833
#
# Capability posiblities:
#    g711Alaw
#    g711Mulaw
#    gsm
#    gsmEFR
#    g723_5_3k
#    g723_6_3k
#    g729a
#    g729ab
#    t38UDPFax
#
#    Note: if you want to run the demo with coder g729 use:
#    g729a for running with VAD disable
#    and 729ab for running with VAD enable
#
# Caution:
```

```
#      If capability is g711Alaw /Mulaw  ==> FramesPerPkt = 10,20,30.
#                                             G711 frame per packet defines the packet size in
#                                             milliseconds
#      If capability is g723_5_3k / 6_3k ==> FramesPerPkt = 1, 2, 3 .
#                                             FrameSize isn't needed, default= 30ms.
#      If capability is gsm               ==> FramesPerPkt = 1, 2, 3 .
#                                             FrameSize isn't needed, default= 20ms.
#      If capability is gsmEFR            ==> FramesPerPkt = 1, 2, 3 .
#                                             FrameSize isn't needed, default= 20ms.
#      If capability is g729a             ==> FramesPerPkt = 3, 4 .
#                                             FrameSize isn't needed, default= 10ms.
#                                             VAD disable, the VAD parameter is ignored
#      If capability is g729ab             ==>FramesPerPkt = 3, 4 .
#                                             FrameSize isn't needed, default= 10ms.
#                                             VAD enable, the VAD parameter is ignored
#
#############################################################################################

ipProtocolName = H323
DTMFmode = inBand
Channel = 1-120
{
   Capability
   {
      TxType = g711Mulaw
      TxFramesPerPkt = 30
      TxVAD = 0
      RxType = g711Mulaw
      RxFramesPerPkt = 30
      RxVAD = 0
   }
   MediaAlarmLostPackets
   {
      Threshold      = 20    # Threshold value
      DebounceOn     = 10000 # Threshold debounce ON
      DebounceOff    = 10000 # Threshold debounce OFF
      Interval       = 1000  # Threshold Time Interval (ms)
      PercentSuccess = 60    # Threshold Success Percent
      PercentFail    = 40    # Threshold Fail Percent
   }

   MediaAlarmJitter
   {
      Threshold        = 60    # Threshold value
      DebounceOn       = 20000 # Threshold debounce ON
      DebounceOff      = 60000 # Threshold debounce OFF
      Interval         = 5000  # Threshold Time Interval (ms)
      PercentSuccess   = 60    # Threshold Success Percent
      PercentFail      = 40    # Threshold Fail Percent
   }

# MediaAlarmResetAlarmState   = 0

}
```

3.2   Compiling and Linking

Compile the project within the following environments:

- Microsoft Visual Studio® 6.0 with Service Pack 5

If you have added or changed files, to compile the project put the files in *C:\Program Files\Dialogic\Demos\IPDemos\IPMediaServer*.

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

intel®

Running the Demo   **4**

This chapter discusses how to run the IP Media Server (Global Call) 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\IPDemos\IPMediaServer\Release\IPMediaServer.exe.* Click OK to run the IP Media Server (Global Call) 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.

Table 1. Command Line Switches

Switch	Action	Default
-c\<filename\>	Configuration file name	-cIPMediaServer.cfg
-e\<encoding type\>	Sets the encoding type: • m – muLaw • a – A-Law	-em
-h or ?	Prints the command syntax to the screen	Off
-n\<*n*\>	Sets the number of channels	The lesser of Voice Devices or IP devices
-q	Activates Quality of Service	Disabled

Example

The following example shows how to launch the demo from a command line with options:

```
IPMediaServer -n64 -cmyconfig.cfg -ea
```

This command launches the demo with 64 channels, using the *myconfig.cfg* configuration file and A-Law encoding.

4.3 Using the Demo

This section describes how to use the IP Media Server (Global Call) demo and contains the following topics:

- Keyboard Commands
- Using the Media Server

4.3.1 Keyboard Commands

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
q or Q or Ctrl+c	Terminates the application

4.3.2 Using the Media Server

The IP Media Server (Global Call) demo allows the caller to interact with a series of voice menus, using the telephone keypad to enter an option. Basic operations include playing a pre-recorded message, recording a new message, and barge-in for CSP. Each menu prompts the caller to select an action by pushing a key. The call state within which the menu is called is indicated by the square brackets. Figure 1 shows the voice prompt flow, grouping the prompts by feature (voice, CSP).

Main Menu [Main_Menu]

1 - Voice Mail
4 - CSP
* - Quit

Voice Mail Menu [Voicemail_Menu]

1 - Record Message
2 - Listen to message from a mailbox
* - Quit

Send Message Prompt [SendMsg_Menu]

Enter Mailbox Number - between 101 - 299
* - Quit

Start Record Prompt [Record_Menu]

2 - Start Record
Press 2 at end of message to stop recording (at end Stop Record Prompt is played)
* - Quit

Stop Record Prompt [StopRec_Menu]

2 - Discard Message and re-record message to same mailbox
3 - Confirm Message [Save_Confirm]
4 - Listen to Message (and replay Stop Record Prompt)
* - Quit without saving

Confirm Message Saved [StopRec_Menu]

1 - Record a message
2 - Listen to a message from a mailbox
* - Quit

Listen to Message Prompt [ListenMsg_Menu]

Enter Mailbox Number - between 101 - 299 (Recorded message is played)
* - Quit

Stop Listen Prompt [Listening]

2 - Discard message and quit
* - Save message and quit

CSP Prompt [CSP_Menu]

Say something to barge in (new file will be created with barged in message)
* - Quit

Figure 1. Demo Voice Menu Flowchart

Main Menu
Select one of the following:
1 - Voice Mail
4 - CSP
* - Quit

CSP Prompt
Say something to barge in
* - Quit

Voice Mail Prompt
1 - Record Message
2 - Listen to message
    from a mailbox
* - Quit

Send Message Prompt
Enter Mailbox
    Number
* - Quit

Listen Message Prompt
Enter Mailbox
    Number (Message
    is played)
* - Quit

Mailbox number

Mailbox number

Start Record Prompt
2 - Start/Stop Record
* - Quit

Stop Listen Prompt
2 - Discard message
    and quit
* - Save message and
    quit

Stop Record Prompt
2 - Discard message and
    re-record message to
    same mailbox
3 - Confirm message
4 - Replay message
* - Quit

Confirm Message Saved Prompt
Confirm message
Return to Voice Mail prompt

Replay Message
Replay Stop Record
Prompt

intel®

4.4   Stopping the Demo

The IP Media Server (Global Call) demo runs until it is terminated. Press "q" or "Q" or "Ctrl+C" to terminate the demo application.

intel

Demo Details 5

This chapter discusses the IP Media Server (Global Call) demo in more detail. It contains the following topics:

5.1 Files Used by the Demo

This section lists the files used by the IP Media Server (Global Call) demo. It contains the following information

- Demo Source Code 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\IPDemo

Table 3. Files Used by the IP Media Server (Global Call) Demo

Directory	File Name	Purpose
IPMediaServer	CConfig.cpp	Implements the operations of the Configuration class
IPMediaServer	CConfig.h	Function prototype for config.cpp
IPMediaServer	CEventRouter.cpp	Implements the operations of the EventRouter class
IPMediaServer	CEventRouter.h	Function prototype for ceventrouter.cpp
IPMediaServer	IPMediaServer.cfg	Demo configuration file
IPMediaServer	IPMediaServer.dsp	Visual C++ project file
IPMediaServer	IPMediaServer.dsw	Visual C++ project workspace
IPMediaServer	main.cpp	Contains the main function and the WaitForKey
IPMediaServer	main.h	Function prototype for main.cpp
IPMediaServer\release	unavConf.vox	Voice file
IPMediaServer\Release	cspPrompt.vox	Voice file

Table 3. Files Used by the IP Media Server (Global Call) Demo (Continued)

Directory	File Name	Purpose
IPMediaServer\Release	errorInput.vox	Voice file
IPMediaServer\Release	IPMediaServer.cfg	Demo configuration file
IPMediaServer\Release	IPMediaServer.exe	Demo executable
IPMediaServer\Release	listenMenu.vox	Voice file
IPMediaServer\Release	mainMenu.vox	Voice file
IPMediaServer\Release	savecCnfirm.vox	Voice file
IPMediaServer\Release	sendfaxMenu.vox	Voice file
IPMediaServer\Release	sendMsg.vox	Voice file
IPMediaServer\Release	startRec.vox	Voice file
IPMediaServer\Release	stopListen.vox	Voice file
IPMediaServer\Release	stopRec.vox	Voice file
IPMediaServer\Release	thankYou.vox	Voice file
IPMediaServer\Release	unavCsp.vox	Voice file
IPMediaServer\Release	unavFax.vox	Voice file
IPMediaServer\Release	unavMenu.vox	Voice file
IPMediaServer\Release	voicemailMenu.vox	Voice file
Modules\common	Defines.h	Definitions and structures for the demo
Modules\common	Interfaces.h	Interfaces used in the demo
Modules\common	Parameters.h	Parameters used in the demo
Modules\IPModule	CIPDevice.cpp	Implements the operations of the CIPDevice class
Modules\IPModule	CIPDevice.h	Function prototype for cipdevice.cpp
Modules\IPModule	CIPMBoard.cpp	Implements the operations of the CIPMBoard class
Modules\IPModule	CIPMBoard.h	Function prototype for cipmboard.cpp
Modules\IPModule	CIPModule.cpp	Implements the operations of the CIPModule class
Modules\IPModule	CIPModule.h	Function prototype for cipmodule.cpp
Modules\IPModule	CIPStateMachine.cpp	Implements the operations of the CIPStateMachine class
Modules\IPModule	CIPStateMachine.h	Function prototype for cipstatemachine.cpp
Modules\IPModule	CIPTBoard.cpp	Implements the operations of the CIPTBoard class
Modules\IPModule	CIPTBoard.h	Function prototype for ciptboard.cpp
Modules\IPModule	CMediaAlarms.cpp	Implements the operations of the CMediaAlarms class
Modules\IPModule	CMediaAlarms.h	Function prototype for cmediaalarms.cpp
Modules\IPModule	IPModule.dsp	Visual C++ project file
Modules\IPModule\ Release	IPModule.lib	Compiled IP Module library

Table 3. Files Used by the IP Media Server (Global Call) Demo (Continued)

Directory	File Name	Purpose
Modules\VoiceModule	CCSPDevice.cpp	Implements the operations of the CCSPDevice class
Modules\VoiceModule	CCSPDevice.h	Function prototype for ccdspdevice.cpp
Modules\VoiceModule	CMailBoxBtil.cpp	Implements the operations of the CMailBoxUtil class
Modules\VoiceModule	CMailBoxUtil.h	Function prototype for cmailboxutil.cpp
Modules\VoiceModule	CVoiceBoard.cpp	Implements the operations of the voiceBoard class
Modules\VoiceModule	CVoiceBoard.h	Function prototype for cvoiceboard.cpp
Modules\VoiceModule	CVoiceDevice.cpp	Implements the operations of the CVoiceDevice class
Modules\VoiceModule	CVoiceDevice.h	Function prototype for cvoicedevice.cpp
Modules\VoiceModule	CVoiceModule.cpp	Implements the operations of the CVoiceModule class
Modules\VoiceModule	CVoiceModule.h	Function prototype for cvoicemodule.cpp
Modules\VoiceModule	CVoiceStateMachine.cpp	Implements the operations of the CVoiceStateMachine class
Modules\VoiceModule	CVoiceStateMachine.h	Function prototype for cvoicestatemachine.cpp
Modules\VoiceModule	VoiceModule.dsp	Visual C++ project file
Modules\VoiceModule\Release	VoiceModule.lib	Compiled Voice Module library

5.1.2   PDL Files

In Windows, the source code files listed in Table 4 are located in:
C:\Program Files\Dialogic\Demos\IPDemo\Shared

Table 4. PDL Files Used by the IP Media Server (Global Call) Demo

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

Table 4. PDL Files Used by the IP Media Server (Global Call) Demo (Continued)

Directory	File Name	Purpose
pdl_win	pdl_win.dsp	PDL Visual C project file
pdl_win	pdl_win.dsw	PDL Visual C project workspace
pdl_win\release	psl_win.lib	Compiled PDL library

5.2 Programming Model

This section describes the IP Media Server (Global Call) demo architecture. It contains the following topics:

- Module Structure
- EventRouter
- Interfaces
- Classes

5.2.1 Module Structure

The IP Media Server (Global Call) demo uses a modular architecture, in which each technology (IP, voice, CSP) is wrapped inside a module so that a particular technology can be easily added or removed.

The system contains two modules:

- IP module that serves as the front end to communicate with the IP network
- Voice module to provide voice service

The system also contains a software component, the EventRouter, to connect the modules. The basic architecture of the system is illustrated in Figure 2.

Figure 2. IP Media Server (Global Call) Demo Architecture
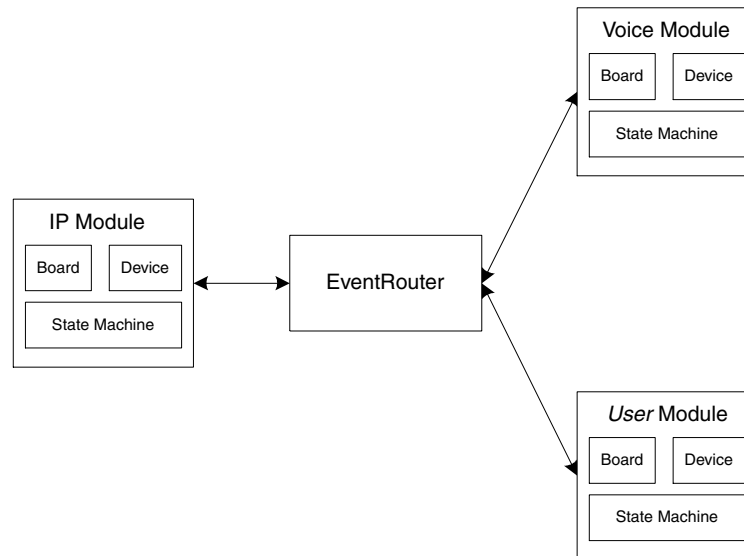
Each module is composed of four elements:

- Board
- Device
- State Machine (call control)
- Wrapper

The Wrapper acts like a manager, receiving requests from the EventRouter and distributing the request to the boards or devices. A device can have one or multiple state machines attached to it. Each state machine represents one call.

5.2.2   EventRouter

The EventRouter is responsible for communicating with the modules. It does the following:

- Maintains routing tables
- Retrieves event data from the SRL and routes it to a module for processing
- Forwards event process result to another module if so requested.

Figure 3. EventRouter
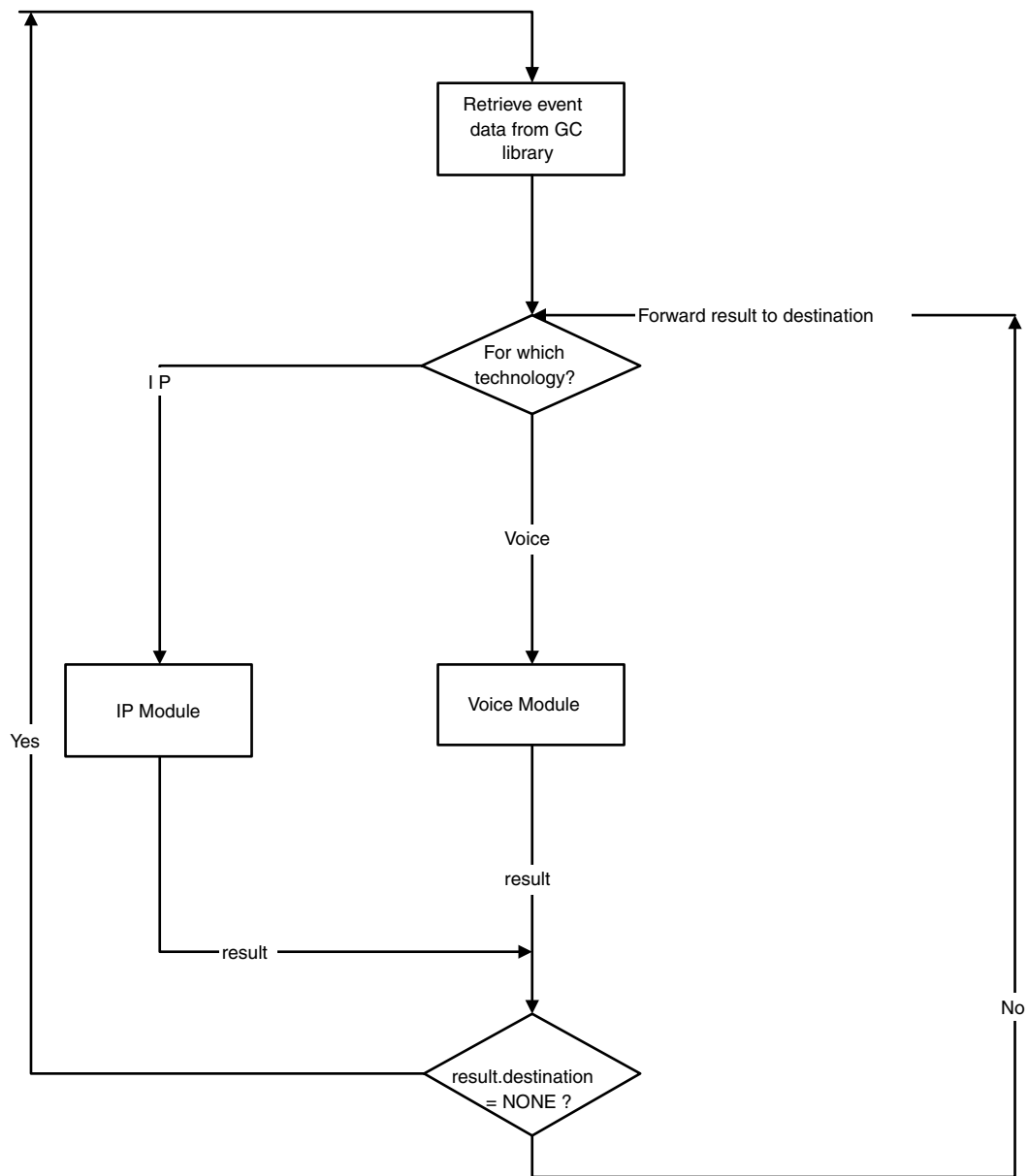
5.2.3   Interfaces

The modular architecture implements a unified interface that allows replacement of modules by including new header files and adjusting routing statements.

A module is treated by the EventRouter as a block box. It has three types of APIs:

Initialization
- **Init( ) –** for initializing a module
- **GetNumOfDevices( ) –** returns the number of devices available to the application
- **GetDeviceHandle( ) –** returns a device handle
- **GetDeviceXmitSlot( ) –** returns a device transmit timeslot
- **SetDeviceReceiveSlot( ) –** sets a device receive timeslot

Runtime
- **ProcessEvent( ) –** processes event data

Termination
- **Exit( ) –** exits a module

5.2.4   Classes

This section describes the classes contained in the demo and within each module. Each module contains three classes: board, device, and state machine.

Table 5. Application Classes

Class Name	Interface
CConfig • Responsible for parsing configuration file and populating configuration variables in the program.	**ParseConfigFile( )** • Parses configuration file
	ChannelNumber( ) • Gets the number of channels that are the minimum of the number of voice lines and the number of IP lines
	IPParms( ) • Returns configured IP parameters
	VoiceParms( ) • Returns configured voice parameters
CEventRouter Responsible for connecting all the modules. It does the following: • Maintains routing tables • Retrieves event data from the SRL and routes them to modules • Gets event process results from the modules and routes them to other modules if the destination is not NONE	**Init( )** • Initializes the IP module, voice module, and fax module • Builds event routing tables
	Exit( ) • Terminates IP module, voice module, and fax module
	ProcessEvent( ) • Retrieves event data from Global Call and routes them to a module for processing • Routes processing result to other module(s) for further processing.

Table 6. IP Module Classes

Category	Class Name	Interface
Module	**CIPModule** • Interacts with CIPBoard and CIPDevice. • Exports IP module functions to CEventRouter. **Note:** The public functions in this class can be accessed by classes outside the IP module.	**Init( )** • Starts host based IP protocol • Creates and initializes IPT (virtual) board objects • Creates and initializes IPM (media) board objects • Creates and initializes IP device objects
		Exit( ) • Terminates and closes IPT boards • Terminates and closes IPM boards • Terminates and closes IP devices
		ProcessEvent( ) • Receives event data from the event router and distributes them, based on handles, to IPT boards, IPM boards or IP devices
		GetNumOfDevices( ) • Returns number of IP devices engaging in the communication
		GetDeviceHandle( ) • Returns an IP device handle
		GetDeviceXmitSlot( ) • Returns the transmit timeslot of an IP device
		SetDeviceReceiveSlot( ) • Sets the receiving timeslot of an IP device

Table 6. IP Module Classes (Continued)

Category	Class Name	Interface
Board	**CIPTBoard** • Interacts with the Global Call library to handle IP virtual boards **Note:** The public function in this class should only be accessed by classes inside the IP module.	**Init( )** • Opens IPT (virtual) board and sets board parameters
		Exit( ) • Closes IPT (virtual) Board
		ProcessEvent( ) • Processes IPT (virtual) board events
		GetNumOfDevicesOnBoard( ) • Returns the number of IPT devices (signaling devices) on the board
	CIPMBoard • Interacts with the Global Call library to handle IP media boards **Note:** The public functions in this class should only be accessed by classes inside the IP module.	**Init( )** • Opens IP media boards
		Exit( ) • Closes IP media boards
		ProcessEvent( ) • Processes IP media board events
		GetNumOfDevices( ) • Returns number of IPM devices on the board

Table 6. IP Module Classes (Continued)

Category	Class Name	Interface
Device	**CIPDevice** • Handles IP device operations, such as making/dropping calls, sending H.323 messages, making timeslot connections, etc. • Holds CIPStateMachine (or call control) **Note:** The public functions in this class should only be accessed by classes inside the IP module.	**Init( )** • Opens an IP device. If successful, creates an IP state machine. If QoS is enabled, creates a media alarm object.
		Exit( ) • Deletes IP state machine • Deletes media alarm object • Closes IP devices
		ProcessEvent( ) • Processes IP device events
		Connect( ) • Allows IP device to listen to its receiving timeslot
		Disconnect( ) • Allows IP device to unlisten to its receiving timeslot
		GetDeviceHandle( ) • Returns IP device handle
		GetXmitSlot( ) • Returns IP device transmit timeslot
		SetDeviceReceiveSlot( ) • Sets IP device receiving timeslot
		SetFaxHandle( ) • Sets fax device handle associated with the device
		GetFaxHandle( ) • Returns fax device handle associated with the device
State Machine	**CIPStateMachine** • Handles IP events and maintains IP state machine(s) **Note:** The public functions in this class should only be accessed by the classes inside the IP module.	**Init( )** • Initializes the IP state machine
		Exit( ) • Terminates the IP state machine
		ProcessEvent( ) • Processes IP call events

Table 7. Voice Module Classes

Category	Class Name	Interface
Module	**CVoiceModule** • Interacts with CVoiceBoard and CVoiceDevice. • Exports Voice module functions to CEventRouter. **Note:** The public functions in this class can be accessed by classes outside the Voice module.	**Init( )** • Opens and initializes voice board objects • Opens and initializes voice device objects **Exit( )** • Terminates and closes voice boards • Terminates and closes voice devices **ProcessEvent( )** • Receives event data from the event router and distributes them, based on devicehandles, to voice boards or voice devices for processing
Board	**CVoiceBoard** • Interacts with the voice library to handle voice boards **Note:** The public function in this class should only be accessed by classes inside the Voice module.	**Init( )** • Opens voice board **Exit( )** • Closes voice board **ProcessEvent( )** • Processes voice board events

Table 7. Voice Module Classes (Continued)

| Category	Class Name	Interface
Device	**CVoiceDevice** • Handles voice device operations, such as playing, recording, tone detection, and tone generation • Holds CVoiceStateMachine **Note:** The public functions in this class should only be accessed by classes inside the Voice module.	**Init( )** • Opens the voice device. If successful, creates voice state machine. **Exit( )** • Deletes the voice state machine from the voice device • Closes voice device **ProcessEvent( )** • Processes voice device events **Connect( )** • Allows IP device to listen to its receiving timeslot **Play( )** • Plays a vox file **OnPlayComplete( )** • Replays a vox file when it has finished playing **Record( )** • Records voice to a vox file **GetDigits( )** • Retrieves DTMF digits **GetDigitCount( )** • Returns the number of retrieved DTMF digits **GetDigitString( )** • Returns retrieved DTMF string **Connect( )** • Allows the voice device to listen to its receiving timeslot **Disconnect( )** • Allows voice device to unlisten to its receiving timeslot **GetStoppedReason( )** • Tells why a played voice file stops **GetDeviceHandle( )** • Returns voice device handle **GetXmitSlot( )** • Returns voice device transmit timeslot **SetReceiveSlot( )** • Sets voice device receiving timeslot

Table 7. Voice Module Classes (Continued)

Category	Class Name	Interface
State Machine	**CVoiceStateMachine** • Handles voice events and maintains voice state machine(s) **Note:** The public functions in this class should only be accessed by the classes inside the voice module.	**Init( )** • Opens vox files • Creates mailbox utility object that manages mailboxes
		Exit( ) • Closes vox files • Deletes mailbox utility object that manages mailboxes
		ProcessEvent( ) • Processes voice events
Misc	**CMailBoxUtil** • Provides utility function to manage mailboxes **Note:** The public functions in this class should only be accessed by the classes inside the voice module	**InitMailBoxes( )** • Initializes mailboxes
		CheckAndConvertDigits( ) • Checks if the extension number is in allowed range. If it is, converts it into an integer.
		GetMailBox( ) • Checks if the mailbox is ready to be used. If it is, gets the mailbox.
		CreateMailBoxFileName( ) • Creates a filename for the mailbox
		FreeMailBox( ) • Frees the mailbox for future use

5.3 Threads

The IP Media Server (Global Call) demo operates with two threads, as shown in Figure 4.

Figure 4. IP Media Server (Global Call) Demo Threads
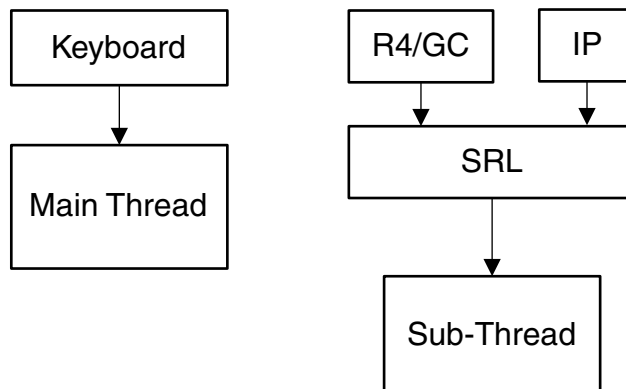
The threads are created as follows:

1. The first (main) thread is created by the demo application to get the keyboard input.

2. The second thread is an SRL thread, created as a result of the demo application calling **PDLsr_enblhdlr( )**. All Global Call events are received through the SRL.

5.4   Initialization

This section describes the demo initialization as shown in Figure 5.

A system is started in the following sequence:

1. The application creates **CConfig( )** to parse the configuration file.

2. The application creates **CEventRouter( )** to start the Event Router, which, in turn, starts the IP module and the Voice module. When a module is started, it initializes its boards, devices and state machines.

After all the modules are started, the EventRouter starts to build a static routing table that maps voice devices to IP devices. It is important that each IP device has a dedicated voice resource, so that when an IP channel is connected (an incoming call is answered) the user can get a voice prompt immediately.

If the initialization should fails, the application shuts down. The shutdown sequence is the reverse of the initialization sequence.

Figure 5. System Initialization
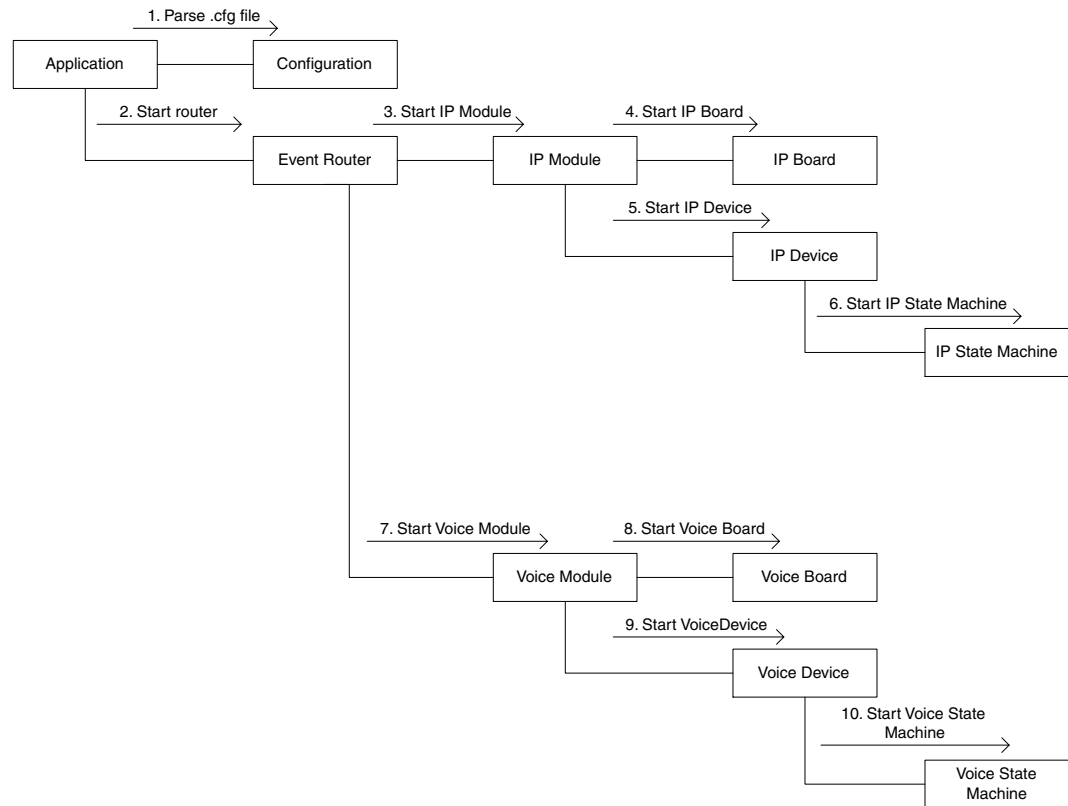
5.5   Event Handling

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

- Event Mechanism
- Handling Keyboard Input Events
- Handling SRL Events
- Handling Application Exit Events

5.5.1   Event Mechanism

The IP Media Server (Global Call) 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 **CallbackHdlr( )** function for handling.

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

5.5.2   Handling Keyboard Input Events

There is an endless loop **{while(0)}** 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( )**.

5.5.3   Handling SRL Events

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

1. Get METAEVENT by calling **gc_GetMetaEvent( )**.

2. Get channel ID through device handle to channel mapping.

3. Get device type through device handle to type mapping.

4. Route meta event to the module specified by device type.

5.5.4   Handling Application Exit Events

Normal application exit events, such as pressing either q or Q, or Ctrl+C, don't enter the SRL. The **main( )** function calls **PDLSetApplicationExitPath( )** before initialization. In Windows, this function enables the detection of CTRL_CLOSE_EVENT (closing the window).

5.5.5   TSUsrEvent Structure

The TSUsrEvent structure is used by the demo modules to return event processing results to the event router. It has five fields:

Event
    The name of a user defined event, such as USR_CONNECTED, etc.

Destination
    The name of the module that this event is destined for. A destination can be IP, VOICE, or NONE.

lineDevice
    The device handle in this module. It will be later used by the Event Router as an index to find its counterpart in the destination module.

xmitSlot
    An integer normally to be filled with with a time slot

neighborDevice
    neighbor device handle

dialString
    A char string normally to be filled with a DTMF dialing string.

The TSUserEvt structure is defined as follows:

```
typedef struct
{
  int        event;
  TDeviceType destination;
  int        lineDevice;
  long       xmitSlot;
  long       neighborDevice;
  char       dialString[MAX_STRING_LENGTH]}
TSUserEvent;
```

intel®

Glossary

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.

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: 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

SCbus: The standard bus for communication within a SCSA node. The architecture of the 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

intel.

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.

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

intel.

Index

Symbols

{while(0)} 40

A

Application Classes   31

C

CallbackHdlr( ) 39
CConfig( ) 38
CEventRouter( ) 38
ChannelNumber( ) 31
CheckAndConvertDigits( ) 37
Classes 31
Command Line Switches   19
Compiling and Linking 17
Connect( ) 34, 36
CreateMailBoxFileName( ) 37

D

Demo Description 11
Demo Details 25
Demo Options 19
Demo Source Code Files 25
Demo Voice Menu Flowchart 22
Disconnect( ) 34, 36

E

Editing Configuration Files 15
Editing the ipmediaserver.cfg Configuration File 15
Event Handling 39
Event Mechanism 39
EventRouter 29, 30
Exit( ) 31, 32, 33, 34, 35, 36, 37

F

File Location 15
Files Used by the Demo 25
Files Used by the IP Media Server (Global Call) Demo   25

FreeMailBox( ) 37

G

gc_GetMetaEvent( ) 40
GetDeviceHandle( ) 31, 32, 34, 36
GetDeviceXmitSlot( ) 31, 32
GetDigitCount( ) 36
GetDigits( ) 36
GetDigitString( ) 36
GetFaxHandle( ) 34
GetMailBox( ) 37
GetNumOfDevices( ) 31, 32, 33
GetNumOfDevicesOnBoard( ) 33
GetStoppedReason( ) 36
GetXmitSlot( ) 34, 36

H

Handling Application Exit Events 40
Handling Keyboard Input Events 40
Handling SRL Events 40
Hardware Requirements 13

I

Init( ) 31, 32, 33, 34, 35, 36, 37
Initialization 38
InitMailBoxes( ) 37
Interfaces 30
IP Media Server (Global Call) Demo Architecture 29
IP Media Server (Global Call) Demo Threads 37
IP Module Classes   32
IPParms( ) 31

K

Keyboard Commands 20

M

main( ) 39, 40
Module Structure 28