IP Media Server (Global Call)

Demo Guide

August 2005
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision History</td>
<td>7</td>
</tr>
<tr>
<td>About This Publication</td>
<td>9</td>
</tr>
<tr>
<td>1 Demo Description</td>
<td>11</td>
</tr>
<tr>
<td>2 System Requirements</td>
<td>13</td>
</tr>
<tr>
<td>2.1 Hardware Requirements</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Software Requirements</td>
<td>13</td>
</tr>
<tr>
<td>3 Preparing to Run the Demo</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Editing Configuration Files</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1 Configuration File Location</td>
<td>15</td>
</tr>
<tr>
<td>3.1.2 Editing the IPMediaServer.cfg Configuration File</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Compiling and Linking</td>
<td>18</td>
</tr>
<tr>
<td>4 Running the Demo</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Starting the Demo</td>
<td>21</td>
</tr>
<tr>
<td>4.2 Demo Options</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Using the Demo</td>
<td>22</td>
</tr>
<tr>
<td>4.3.1 Keyboard Commands</td>
<td>22</td>
</tr>
<tr>
<td>4.3.2 Using the Media Server</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Stopping the Demo</td>
<td>25</td>
</tr>
<tr>
<td>5 Demo Details</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Files Used by the Demo</td>
<td>27</td>
</tr>
<tr>
<td>5.1.1 Demo Source Code Files</td>
<td>27</td>
</tr>
<tr>
<td>5.1.2 PDL Files</td>
<td>29</td>
</tr>
<tr>
<td>5.2 Programming Model</td>
<td>30</td>
</tr>
<tr>
<td>5.2.1 Module Structure</td>
<td>30</td>
</tr>
<tr>
<td>5.2.2 EventRouter</td>
<td>31</td>
</tr>
<tr>
<td>5.2.3 Interfaces</td>
<td>32</td>
</tr>
<tr>
<td>5.2.4 Classes</td>
<td>33</td>
</tr>
<tr>
<td>5.3 Threads</td>
<td>39</td>
</tr>
<tr>
<td>5.4 Initialization</td>
<td>40</td>
</tr>
<tr>
<td>5.5 Event Handling</td>
<td>42</td>
</tr>
<tr>
<td>5.5.1 Event Mechanism</td>
<td>42</td>
</tr>
<tr>
<td>5.5.2 Handling Keyboard Input Events</td>
<td>42</td>
</tr>
<tr>
<td>5.5.3 Handling SRL Events</td>
<td>42</td>
</tr>
<tr>
<td>5.5.4 Handling Application Exit Events</td>
<td>42</td>
</tr>
<tr>
<td>5.5.5 TSUsrEvent Structure</td>
<td>43</td>
</tr>
<tr>
<td>Glossary</td>
<td>45</td>
</tr>
<tr>
<td>Index</td>
<td>49</td>
</tr>
</tbody>
</table>
Contents

Figures

1 Demo Voice Menu Flowchart .......................................................... 24
2 IP Media Server (Global Call) Demo Architecture .................................. 31
3 EventRouter .............................................................................. 32
4 IP Media Server (Global Call) Demo Threads ........................................... 39
5 System Initialization ........................................................................ 41
Contents

Tables

1 Quality of Service Parameter Defaults for DM/IP Series Boards ........................................... 16
2 Quality of Service Parameter Defaults for IPT Series Boards .................................................. 17
3 Command Line Switches ............................................................................................................. 21
4 Runtime Keyboard Commands .................................................................................................. 22
5 Files in IPMediaServer Folder Used by the IP Media Server (Global Call) Demo .......................... 27
6 Files in Modules Folder Used by the IP Media Server (Global Call) Demo ............................... 28
7 PDL Files Used by the IP Media Server (Global Call) Demo ..................................................... 30
8 Application Classes .................................................................................................................. 33
9 IP Module Classes ................................................................................................................... 34
10 Voice Module Classes .............................................................................................................. 37
This revision history summarizes the changes made in each published version of this document.

<table>
<thead>
<tr>
<th>Document No.</th>
<th>Publication Date</th>
<th>Description of Revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>05-2065-002</td>
<td>November 2003</td>
<td>Removed all references to fax, which is not supported in System Release version of demo.</td>
</tr>
<tr>
<td>05-2065-001</td>
<td>September 2003</td>
<td>Initial version of document.</td>
</tr>
</tbody>
</table>
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 provided with Intel® Dialogic® System Release 6.1. 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 either the Windows® or Linux® 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 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.

Related Information

See the following for more information:

• the Release Guide for your Intel Dialogic System Release
• the Configuration Guide for your Intel NetStructure® DM/IP series or IPT series board
• Global Call IP Technology Guide
• Voice API Programming Guide
• Voice API Library Reference
• Standard Runtime Library API Programming Guide
• Standard Runtime Library API Library Reference
• http://developer.intel.com/design/telecom/support/ (for technical support)
• http://www.intel.com/design/network/products/telecom (for product information)
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.

Note: The IP Media Server (Global Call) demo is limited to a maximum of four simultaneous channels. Additionally, the channel numbers used must be below 120; the demo will fail to run if you attempt to use a channel number higher than 120.

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 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` or `pdl_linux` directories directory.
Demo Description
This chapter discusses the system requirements for running the IP Media Server (Global Call) demo. It contains the following topics:

- Hardware Requirements ................................................. 13
- Software Requirements .................................................. 13

2.1 Hardware Requirements

To run the IP Media Server (Global Call) demo, you need:

- One of the following:
  - Intel NetStructure® DM/IP 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 the system release you are using.

2.2 Software Requirements

To run the IP Media Server (Global Call) demo as documented in this guide, you need the Intel® Dialogic® System Software 6.1 or later. 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 18 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

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

- Editing Configuration Files ........................................................................ 15
- Compiling and Linking ................................................................................ 18

3.1 Editing Configuration Files

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

- Configuration File Location
- Editing the IPMediaServer.cfg Configuration File

3.1.1 Configuration 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 the following location:

Windows

$(INTEL_DIALOOGIC_DIR)\Demos\IPMediaServer\Release

Linux

$(INTEL_DIALOOGIC_DIR)/demos/IPMediaServer/Release

where $(INTEL_DIALOOGIC_DIR) identifies the base installation directory for the software.

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. Possible values are:

- H323
- SIP
- both

DTMFmode

Specifies how DTMF tones are transmitted. Possible values are:

- OutofBand – usually used with low bandwidth coders, such as GSM

Note: OutofBand is used for H.323 only.
Preparing to Run the Demo

- InBand – usually used with G.711 coders
- RFC2833

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 – the number of frames per packet for the selected Tx coder
- TxVAD – specifies if VAD is active for the selected Tx coder
- RxType – the receive voice coder
- RxFramesPerPkt – the number of frames per packet 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 the Sample Configuration File, below, for the correct syntax for all the parameters.
- RxVAD – specifies if VAD is active for the selected Rx coder

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 crosses a predefined threshold. A success occurs when the result of a measurement of a QoS parameter does 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 (supported for IPT series boards only)
- 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:

Table 1. Quality of Service Parameter Defaults for DM/IP Series Boards

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

### Table 2. Quality of Service Parameter Defaults for IPT Series Boards

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

Notes:
1. Units for Fault Threshold are different for different QoS Types. See unit indications in table cells.

### Sample Configuration File

```plaintext
################################################################################################
# IP Protocol :
#    The IP Protocol used for opening the IP Line devices, values: H323, SIP, both
#
# DTMFmode
#       possible options:
#            OutOfBand, inband, rfc2833
#
# Capability possible:
#    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.
#    711 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 = g711Alaw
    # TxFramesPerPkt = 30
```

IP Media Server (Global Call) Demo Guide — August 2005
Preparing to Run the Demo

```plaintext
#   TxVAD = 0
#   RxType = g711Alaw
#   RxFramesPerPkt = 30
#   RxVAD = 0

TxType = g711Mulaw
TxFramesPerPkt = 20
TxVAD = 0
RxType = g711Mulaw
RxFramesPerPkt = 20
RxVAD = 0

#   TxType = t38UDPFax

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 IP Media Server (Global Call) demo project within one of the following environments:

Windows

To compile the IP Media Server (Global Call) demo on a Windows system, use Microsoft® Visual Studio® 6.0 with Service Pack 5.

Set IPMediaServer as the active project and build.
**Linux**

To compile the IP Media Server (Global Call) demo on a Linux system, use gcc version 3.2.3.

To compile the entire project, go to the directory

$\texttt{(INTEL\_DIALOGIC\_DIR)/demos/IPMediaServer}$ and issue the commands:

```
make clean
make
```

To compile an individual module, go to the specific module directory (for example, $\texttt{(INTEL\_DIALOGIC\_DIR)/demos/IPMediaServer/Modules/FaxModule}$ for the fax module), and issue the commands:

```
make clean
make
```
Preparing to Run the Demo
Running the Demo

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

- Starting the Demo .......................................................... 21
- Demo Options ................................................................. 21
- Using the Demo ............................................................... 22
- Stopping the Demo ........................................................... 25

4.1 Starting the Demo

The demo executable file for the IP Media Server (Global Call) demo can be started as follows:

**Windows**

From a command prompt window, change to the directory:

```plaintext
$(INTEL_DIALOGIC_DIR)\Demos\IPMediaServer\Release
```

Type `IPMediaServer` to run the IP Media Server (Global Call) demo using the default settings.

**Linux**

Change to the directory:

```plaintext
$(INTEL_DIALOGIC_DIR)/demos/IPMediaServer/Release
```

Type `IPMediaServer` 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 3.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Action</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c&lt;filename&gt;</code></td>
<td>Configuration file name</td>
<td><code>-cIPMediaServer.cfg</code></td>
</tr>
<tr>
<td><code>-e&lt;encoding type&gt;</code></td>
<td>Sets the encoding type:</td>
<td><code>-em</code></td>
</tr>
<tr>
<td></td>
<td>• m – mu-law</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• a – A-law</td>
<td></td>
</tr>
<tr>
<td><code>-h or ?</code></td>
<td>Prints the command syntax to the screen</td>
<td>Off</td>
</tr>
</tbody>
</table>
### Running the Demo

#### Example

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

```
IPMediaServer -n64 -c myconfig.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 4.

#### Table 4. Runtime Keyboard Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>q or Q or Ctrl+c</td>
<td>Terminates the application</td>
</tr>
</tbody>
</table>

#### 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
- 3 - Conferencing (not supported in HMP 1.x)
- 4 - CSP Barge-in
- * - Quit
Running the Demo

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) or press any DTMF.
Figure 1. Demo Voice Menu Flowchart

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

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

CSP Prompt
Say something to barge in
* - Quit

Send Message Prompt
Enter Mailbox Number
* - Quit

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

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

Start Record Prompt
2 - Start/Stop Record
* - 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
4.4 Stopping the Demo

The IP Media Server (Global Call) demo runs until it is terminated. Press “q” or “Q” to terminate the demo application.
Demo Details

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

- Files Used by the Demo ................................................................. 27
- Programming Model ................................................................. 30
- Threads .......................................................... 39
- Initialization ................................................................. 40
- Event Handling ................................................................. 42

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 IP Media Server (Global Call) demo files listed in Table 5 are located within:
$(INTEL_DIALOGIC_DIR)\Demos\IPMediaServer

In Linux, the IP Media Server (Global Call) demo files listed in Table 5 are located within:
$(INTEL_DIALOGIC_DIR)/demos/IPMediaServer

Table 5. Files in IPMediaServer Folder Used by the IP Media Server (Global Call) Demo

<table>
<thead>
<tr>
<th>Sub-Directory (if any)</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CConfig.cpp</td>
<td>Implements the operations of the Configuration class</td>
</tr>
<tr>
<td></td>
<td>CConfig.h</td>
<td>Function prototype for config.cpp</td>
</tr>
<tr>
<td></td>
<td>CEventRouter.cpp</td>
<td>Implements the operations of the EventRouter class</td>
</tr>
<tr>
<td></td>
<td>CEventRouter.h</td>
<td>Function prototype for ceventrouter.cpp</td>
</tr>
<tr>
<td></td>
<td>IPMediaServer.dsp</td>
<td>Visual C++ project file</td>
</tr>
<tr>
<td></td>
<td>IPMediaServer.dsw</td>
<td>Visual C++ project workspace</td>
</tr>
<tr>
<td></td>
<td>main.cpp</td>
<td>Contains the main function and the WaitForKey</td>
</tr>
<tr>
<td></td>
<td>main.h</td>
<td>Function prototype for main.cpp</td>
</tr>
<tr>
<td></td>
<td>makefile</td>
<td>Top-level make file (Linux only)</td>
</tr>
</tbody>
</table>
Table 5. Files in IPMediaServer Folder Used by the IP Media Server (Global Call) Demo

<table>
<thead>
<tr>
<th>Sub-Directory (if any)</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>unavConf.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>cspPrompt.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>errorInput.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>IPMediaServer.cfg</td>
<td>Demo configuration file</td>
</tr>
<tr>
<td>Release</td>
<td>IPMediaServer.exe</td>
<td>Demo executable</td>
</tr>
<tr>
<td>Release</td>
<td>listenMenu.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>mainMenu.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>savecCnfirm.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>sendflaxMenu.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>sendMsg.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>startRec.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>stopListen.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>stopRec.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>thankyOU.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>unavCsp.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>unavFax.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>unavMenu.vox</td>
<td>Voice file</td>
</tr>
<tr>
<td>Release</td>
<td>voicemailMenu.vox</td>
<td>Voice file</td>
</tr>
</tbody>
</table>

In Windows, the IP Media Server (Global Call) demo files listed in Table 6 are located within:
$\{INTEL_DIALOGIC_DIR\}/Demos/IPMediaServer/Modules

In Linux, the IP Media Server (Global Call) demo files listed in Table 6 are located within:
$\{INTEL_DIALOGIC_DIR\}/demos/IPMediaServer/Modules

Table 6. Files in Modules Folder Used by the IP Media Server (Global Call) Demo

<table>
<thead>
<tr>
<th>Sub-Directory</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>Defines.h</td>
<td>Definitions and structures for the demo</td>
</tr>
<tr>
<td>common</td>
<td>Interfaces.h</td>
<td>Interfaces used in the demo</td>
</tr>
<tr>
<td>common</td>
<td>Parameters.h</td>
<td>Parameters used in the demo</td>
</tr>
<tr>
<td>common</td>
<td>makefile</td>
<td>Module make file (Linux only)</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPDevice.cpp</td>
<td>Implements the operations of the CIPDevice class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPDevice.h</td>
<td>Function prototype for cipdevice.cpp</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPMBoard.cpp</td>
<td>Implements the operations of the CIPMBoard class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPMBoard.h</td>
<td>Function prototype for cipmboard.cpp</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPModule.cpp</td>
<td>Implements the operations of the CIPModule class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPModule.h</td>
<td>Function prototype for cipmodule.cpp</td>
</tr>
</tbody>
</table>
5.1.2 PDL Files

In Windows, the PDL files listed in Table 7 are located within the directory:

\$(INTEL_DIALOGIC_DIR)\Demos\Shared\pdl_win

In Linux, the PDL files listed in Table 7 are located within the directory:

\$(INTEL_DIALOGIC_DIR)/demos/Shared/pdl_Linux

<table>
<thead>
<tr>
<th>Sub-Directory</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPModule</td>
<td>CIPStateMachine.cpp</td>
<td>Implements the operations of the CIPStateMachine class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPStateMachine.h</td>
<td>Function prototype for cipstatemachine.cpp</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPTBoard.cpp</td>
<td>Implements the operations of the CIPTBoard class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CIPTBoard.h</td>
<td>Function prototype for ciptboard.cpp</td>
</tr>
<tr>
<td>IPModule</td>
<td>CMediaAlarms.cpp</td>
<td>Implements the operations of the CMediaAlarms class</td>
</tr>
<tr>
<td>IPModule</td>
<td>CMediaAlarms.h</td>
<td>Function prototype for cmediaalarms.cpp</td>
</tr>
<tr>
<td>IPModule</td>
<td>IPModule.dsp</td>
<td>Visual C++ project file</td>
</tr>
<tr>
<td>IPModule</td>
<td>makefile</td>
<td>Module-level make file (Linux only)</td>
</tr>
<tr>
<td>IPModule/Release or IPModule/Release</td>
<td>IPModule.lib</td>
<td>Compiled IP Module library</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CCSPDevice.cpp</td>
<td>Implements the operations of the CCSPDevice class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CCSPDevice.h</td>
<td>Function prototype for ccspdevice.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CMailBoxUtil.cpp</td>
<td>Implements the operations of the CMailBoxUtil class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CMailBoxUtil.h</td>
<td>Function prototype for cmailboxutil.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceBoard.cpp</td>
<td>Implements the operations of the voiceBoard class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceBoard.h</td>
<td>Function prototype for cvoiceboard.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceDevice.cpp</td>
<td>Implements the operations of the CVoiceDevice class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceDevice.h</td>
<td>Function prototype for cvoicedevice.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceModule.cpp</td>
<td>Implements the operations of the CVoiceModule class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceModule.h</td>
<td>Function prototype for cvoicemodule.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceStateMachine.cpp</td>
<td>Implements the operations of the CVoiceStateMachine class</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>CVoiceStateMachine.h</td>
<td>Function prototype for cvoicestatemachine.cpp</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>VoiceModule.dsp</td>
<td>Visual C++ project file</td>
</tr>
<tr>
<td>VoiceModule</td>
<td>makefile</td>
<td>Module-level make file (Linux only)</td>
</tr>
<tr>
<td>VoiceModule/Release or VoiceModule/Release</td>
<td>VoiceModule.lib</td>
<td>Compiled Voice Module library</td>
</tr>
</tbody>
</table>
Table 7. PDL Files Used by the IP Media Server (Global Call) Demo

<table>
<thead>
<tr>
<th>Sub-Directory (if any)</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iptransport.cpp</td>
<td>PDL IP transport functions</td>
</tr>
<tr>
<td></td>
<td>iptransport.h</td>
<td>Function prototype for iptransport.cpp</td>
</tr>
<tr>
<td></td>
<td>pdl.c</td>
<td>Platform dependency functions</td>
</tr>
<tr>
<td></td>
<td>pdl.h</td>
<td>Function prototype for pdl.c</td>
</tr>
<tr>
<td></td>
<td>pdl.ver</td>
<td>PDL version information</td>
</tr>
<tr>
<td></td>
<td>pdl_win.dsp</td>
<td>PDL Visual C project file</td>
</tr>
<tr>
<td></td>
<td>pdl_win.dsw</td>
<td>PDL Visual C project workspace</td>
</tr>
<tr>
<td></td>
<td>makefile</td>
<td>PDL make file (Linux only)</td>
</tr>
<tr>
<td></td>
<td>release</td>
<td>Compiled PDL library</td>
</tr>
</tbody>
</table>

5.2 Programming Model

This section describes the IP Media Server (Global Call) demo architecture in 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.
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.
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
Demo Details

- `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 8. Application Classes

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>CConfig</td>
<td><code>ParseConfigFile()</code> – Parses configuration file</td>
</tr>
<tr>
<td></td>
<td><code>ChannelNumber()</code> – Gets the number of channels that are the minimum of the number of voice lines and the number of IP lines</td>
</tr>
<tr>
<td></td>
<td><code>IPParms()</code> – Returns configured IP parameters</td>
</tr>
<tr>
<td></td>
<td><code>VoiceParms()</code> – Returns configured voice parameters</td>
</tr>
<tr>
<td>CEventRouter</td>
<td><code>Init()</code> – Initializes the IP module and voice module</td>
</tr>
<tr>
<td></td>
<td><code>Builds event routing tables</code></td>
</tr>
<tr>
<td></td>
<td><code>Exit()</code> – Terminates the voice module and IP module</td>
</tr>
<tr>
<td></td>
<td><code>ProcessEvent()</code> – Retrieves event data from Global Call and routes them to a module for processing</td>
</tr>
<tr>
<td></td>
<td><code>Routes processing result to other module(s) for further processing.</code></td>
</tr>
</tbody>
</table>
### Table 9. IP Module Classes

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

**Board**

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIPTBoard</td>
<td>• Interacts with the Global Call library to handle IP virtual boards</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The public function in this class should only be accessed by classes inside the IP module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Init( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Opens IPT (virtual) board and sets board parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Exit( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Closes IPT (virtual) board</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>ProcessEvent( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes IPT (virtual) board events</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GetNumOfDevicesOnBoard( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns the number of IPT devices (signaling devices) on the board</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CIPMBoard</td>
<td>• Interacts with the Global Call library to handle IP media boards</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The public functions in this class should only be accessed by classes inside the IP module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Init( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Opens IP media boards</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Exit( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Closes IP media boards</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>ProcessEvent( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes IP media board events</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GetNumOfDevices( )</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns number of IPM devices on the board</td>
</tr>
</tbody>
</table>

---

**Table 9. IP Module Classes (Continued)**

---
### Table 9. IP Module Classes (Continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>CIPDevice</td>
<td>Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Opens an IP device. If successful, creates an IP state machine. If QoS is enabled, creates a media alarm object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exit()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Deletes IP state machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Deletes media alarm object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Closes IP devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProcessEvent()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes IP device events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Connect()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows IP device to listen to its receiving timeslot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disconnect()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Allows IP device to unlisten to its receiving timeslot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetDeviceHandle()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns IP device handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetXmitSlot()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns IP device transmit timeslot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SetDeviceReceiveSlot()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sets IP device receiving timeslot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SetFaxHandle()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sets fax device handle associated with the device</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetFaxHandle()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Returns fax device handle associated with the device</td>
</tr>
<tr>
<td>State Machine</td>
<td>CIPStateMachine</td>
<td>Init()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Initializes the IP state machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exit()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Terminates the IP state machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProcessEvent()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes IP call events</td>
</tr>
</tbody>
</table>
### Table 10. Voice Module Classes

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>CVoiceModule</td>
<td>- Interacts with CVoiceBoard and CVoiceDevice.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Exports Voice module functions to CEventRouter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The public functions in this class can be accessed by classes outside the Voice module.</td>
</tr>
<tr>
<td></td>
<td>Init()</td>
<td>- Opens and initializes voice board objects</td>
</tr>
<tr>
<td></td>
<td>Exit()</td>
<td>- Terminates and closes voice boards</td>
</tr>
<tr>
<td></td>
<td>ProcessEvent()</td>
<td>- Receives event data from the event router and distributes them, based on devicehandles, to voice boards or voice devices for processing</td>
</tr>
<tr>
<td>Board</td>
<td>CVoiceBoard</td>
<td>- Interacts with the voice library to handle voice boards</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The public function in this class should only be accessed by classes inside the Voice module.</td>
</tr>
<tr>
<td></td>
<td>Init()</td>
<td>- Opens voice board</td>
</tr>
<tr>
<td></td>
<td>Exit()</td>
<td>- Closes voice board</td>
</tr>
<tr>
<td></td>
<td>ProcessEvent()</td>
<td>- Processes voice board events</td>
</tr>
</tbody>
</table>
Table 10. Voice Module Classes (Continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
</table>
| 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  
|          | GetReceiveSlot( )| • Sets voice device receiving timeslot  
|          |                  |                                                                           


### 5.3 Threads

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

![Diagram of threads](image)

**Figure 4. IP Media Server (Global Call) Demo Threads**

#### Table 10. Voice Module Classes (Continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Class Name</th>
<th>Interface</th>
</tr>
</thead>
</table>
| **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 |
Demo Details

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 IP Media Server (Global Call) 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 fail, the application shuts down. The shutdown sequence is the reverse
of the initialization sequence.
Figure 5. System Initialization

1. Parse .cfg file
2. Start router
3. Start IP Module
4. Start IP Board
5. Start IP Device
6. Start IP State Machine
7. Start Voice Module
8. Start Voice Board
9. Start Voice Device
10. Start Voice State Machine
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
- TSUsrEvent Structure

### 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:

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, 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).
5.5.5 TSUsrEvent Structure

The TSUsrEvent structure is used by the demo modules to return event processing results to the event router.

The TSUserEvt structure is defined as follows:

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

The fields of the TSUserEvt structure are described as follows:

- **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. Possible values are 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 that normally indicates a time slot number

- **neighborDevice**
  - the neighbor device handle

- **dialString**
  - a char string that is normally filled with a DTMF dialing string
Glossary

Codec: see COder/DECoder

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

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

DTMF: Dual-Tone Multi-Frequency

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

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

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

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

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

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

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

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

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

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

IFP: Internet Facsimile Protocol

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

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

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

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

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

ITU: See International Telecommunications Union.

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

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

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

PSTN: see Public Switched Telephone Network

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

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

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

RTCP: Real Time Control Protocol

RTP: Real Time Protocol

SIP: Session Initiation Protocol: an Internet standard specified by the Internet Engineering Task Force (IETF) in RFC 3261. SIP is used to initiate, manage, and terminate interactive sessions between one or more users on the Internet.
**T1:** A digital transmission link with a capacity of 1.544 Mbps used in North America. Typically channeled into 24 digital subscriber level zeros (DS0s), each capable of carrying a single voice conversation or data stream. T1 uses two pairs of twisted pair wires.

**TCP:** see Transmission Control Protocol

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

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

**UDP:** see User Datagram Protocol

**UDPTL:** Facsimile UDP Transport Layer protocol

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

**VAD:** Voice Activity Detection
Index

A
application classes 33
application exit events 42

C
CallbackHdlr() 42
CConfig() 40
CEventRouter() 40
ChannelNumber() 33
CheckAndConvertDigits() 39
classes 33
  application classes 33
  IP module classes 34
command-line switches 21
commands, keyboard 22
compiling and linking 18
configuration files, editing 15
Connect() 36, 38
CreateMailBoxFileName() 39

D
Disconnect() 36, 38

E
editing configuration files 15
event handling 42
  application exit events 42
  keyboard input events 42
  SRL events 42
  TSUsrEvent structure 43
event mechanism 42
EventRouter 31, 32
Exit() 33

F
files, used by demo 27
FreeMailBox() 39

G
gc_GetMetaEvent() 42
GetDeviceXmitSlot() 33, 34
GetDigitCount() 38
GetDigits() 38
GetDigitString() 38
GetFaxHandle() 36
GetMailBox() 39
GetNumOfDevices() 33, 34, 35
GetNumOfDevicesOnBoard() 35
GetStoppedReason() 38
GetXmitSlot() 36, 38

H
hardware requirements 13

I
Init() 32
initialization 40
InitMailBoxes() 39
interfaces 32
IP module classes 34
ipmediaserver.cfg configuration file 15
IPParms() 33

K
keyboard commands 22
keyboard input events, handling 42

L
linking 18

M
main() 42
module structure 30
OnPlayComplete( ) 38

ParseConfigFile( ) 33
PDL files 29
PDLSetApplicationExitPath( ) 42
PDLsr_enbhdlr( ) 42
PDLsr_enblhdlr( ) 40
Play( ) 38
ProcessEvent( ) 33
programming model 30

Record( ) 38
requirements
  hardware 13
  runtime commands, keyboard 22

SetDeviceReceiveSlot( ) 33, 34, 36
SetFaxHandle( ) 36
SetReceiveSlot( ) 38
source code files 27
SRL events, handling 42
starting the demo 21
stopping the demo 25
switches, command line 21
system initialization 41
system requirements 13

threads 39
TSUsrEvent structure 43

using the media server 22

voice menu flowchart 24
Voice Module classes 37
VoiceParms( ) 33