# 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>Purpose</td>
<td>9</td>
</tr>
<tr>
<td>Intended Audience</td>
<td>9</td>
</tr>
<tr>
<td>How to Use This Publication</td>
<td>9</td>
</tr>
<tr>
<td>Related Information</td>
<td>10</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>3.2 Compiling and Linking</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Selecting PCD/FCD Files</td>
<td>17</td>
</tr>
<tr>
<td>4 Running the Demo</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Starting the Demo</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Demo Options</td>
<td>19</td>
</tr>
<tr>
<td>4.3 Using the Demo</td>
<td>20</td>
</tr>
<tr>
<td>4.3.1 Establishing and Terminating a Call</td>
<td>20</td>
</tr>
<tr>
<td>4.3.2 Keyboard Commands</td>
<td>20</td>
</tr>
<tr>
<td>4.4 Stopping the Demo</td>
<td>21</td>
</tr>
<tr>
<td>5 Demo Details</td>
<td>23</td>
</tr>
<tr>
<td>5.1 Files Used by the Demo</td>
<td>23</td>
</tr>
<tr>
<td>5.1.1 Demo Source Code Files</td>
<td>23</td>
</tr>
<tr>
<td>5.1.2 Utility Files</td>
<td>24</td>
</tr>
<tr>
<td>5.1.3 PDL Files</td>
<td>25</td>
</tr>
<tr>
<td>5.2 Programming Model Classes</td>
<td>25</td>
</tr>
<tr>
<td>5.2.1 Class Diagram</td>
<td>25</td>
</tr>
<tr>
<td>5.2.2 Call Class</td>
<td>26</td>
</tr>
<tr>
<td>5.2.3 Configuration Class</td>
<td>27</td>
</tr>
<tr>
<td>5.2.4 IPMediaBoard Class</td>
<td>27</td>
</tr>
<tr>
<td>5.2.5 IPMediaDevice Class</td>
<td>28</td>
</tr>
<tr>
<td>5.2.6 R4Device Class</td>
<td>28</td>
</tr>
<tr>
<td>5.2.7 R4LogicalBoard Class</td>
<td>29</td>
</tr>
<tr>
<td>5.2.8 ResourceManager Class</td>
<td>29</td>
</tr>
<tr>
<td>5.2.9 VoiceBoard Class</td>
<td>30</td>
</tr>
<tr>
<td>5.2.10 VoiceDevice Class</td>
<td>31</td>
</tr>
<tr>
<td>5.3 Threads</td>
<td>31</td>
</tr>
<tr>
<td>5.4 Initialization</td>
<td>32</td>
</tr>
<tr>
<td>5.5 Event Handling</td>
<td>33</td>
</tr>
</tbody>
</table>
Contents

5.5.1 Event Mechanism .............................................................. 33
5.5.2 Handling Keyboard Input Events ......................................... 33
5.5.3 Handling SRL Events .......................................................... 33

6 Demo State Machines .............................................................. 35
6.1 Call State Machine ............................................................... 35
6.1.1 Call State Machine Description ........................................... 35
6.1.2 Call::callNull State ........................................................... 36
6.1.3 Call::callStarted State ....................................................... 37
6.1.4 Call::callProceeding State .................................................. 37
6.1.5 Call::callStopped State ....................................................... 37
6.2 IPMediaDevice State Machine ............................................... 37
6.2.1 IPMediaDevice State Machine Description ............................. 37
6.2.2 IPMediaDevice::mediaNull State ........................................... 38
6.2.3 IPMediaDevice::mediaStarted State ...................................... 38
6.2.4 IPMediaDevice::mediaCall State .......................................... 39
6.2.5 IPMediaDevice::mediaStopped State ..................................... 39

Glossary ................................................................................. 41
Index ...................................................................................... 45
## Figures

1. IP Multicast Server (IPML) Demo System ................................................................. 11
2. IP Multicast Server (IPML) Class Diagram ............................................................ 26
3. Thread Diagram ........................................................................................................ 31
4. IP Multicast Server (IPML) System Initialization .................................................... 32
5. Call State Machine .................................................................................................. 36
6. IPMediaDevice State Machine .................................................................................. 38
## Tables

1. Command Line Switches ................................................................. 19
2. Runtime Keyboard Commands ......................................................... 21
3. Source Files Used by the IP Multicast Server (IPML) Demo .................. 23
4. Utility Files Used by the IP Multicast Server (IPML) Demo ................... 24
5. PDL Files Used by the IP Multicast Server (IPML) Demo - Windows OS .... 25
6. Call Class Attributes ................................................................. 26
7. Configuration Class Attributes ....................................................... 27
8. IPMediaBoard Class Attributes ..................................................... 28
9. IPMediaDevice Class Attributes .................................................... 28
10. R4Device Class Attributes .......................................................... 29
11. R4LogicalBoard Class Attributes .................................................. 29
12. ResourceManager Class Attributes ................................................. 30
13. VoiceBoard Class Attributes ........................................................ 30
14. VoiceDevice Class Attributes ....................................................... 31
Revision History

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

The following topics provide information about this guide:

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

Purpose

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

Intended Audience

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

This information is intended for:

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

How to Use This Publication

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

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

The information in this guide is organized as follows:

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

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

Related Information

See the following for more information:

- The online Release Update for your specific system release for information on problems fixed, known problems and workarounds, and documentation updates.
- Intel DM3 Architecture PCI Products on Windows Configuration Guide
- Global Call IP Technology Guide
- http://developer.intel.com/design/telecom/support/ for technical support
This chapter provides a brief description of the IP Multicast Server (IPML) demonstration program.

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

The basic demo system is shown in Figure 1:

**Figure 1. IP Multicast Server (IPML) Demo System**

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

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

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

The IP Multicast Server (IPML) demo is a cross-OS demo, running under the Windows or Linux environments. Most of the differences in the environments are handled directly by the programming interface and are transparent to the user. Other differences, due to inherent
Differences in the operating systems are handled by the Platform Dependency Library (PDL). For more information about the PDL refer to the source code in the `pdl_win` or `pdl_linux` directories.
This chapter discusses the system requirements for running the IP Multicast Server (IPML) demo. It contains the following topics:

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

## 2.1 Hardware Requirements

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

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

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

## 2.2 Software Requirements

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

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

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

- Editing Configuration Files ............................................................... 15
- Compiling and Linking ................................................................. 17
- Selecting PCD/FCD Files ................................................................. 17

3.1 Editing Configuration Files

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

- `C:\Program Files\dialogic\demo\ipdemo\multicastserver\release\`

**Editing the multicastserver.cfg Configuration File**

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

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

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

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

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

- `g711Alaw`
- `g711Mulaw`
- `gsm`
- `gsmEFR`
- `g7231_5_3k`
- `g7231_6_3k`
- `g729a`
- `g729ab`
Preparing to Run the Demo

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

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

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

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

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

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

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

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

Channel = 2
{
    DestinationIP = 224.0.0.2
    DestinationRTP = 2370
    TransmitFile = MC_file2.vox
    TxCoderType = g711mulaw
    TxCoderFramesPerPkt = 1
    TxCoderFrameSize = 30
    TxCoderVAD = 0
    TxPayload = 0
    TxRedPayload = 0
}
Preparing to Run the Demo

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

3.2 Compiling and Linking

Compile the project within the following environments:

- Visual C++ environment, version 6

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

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

3.3 Selecting PCD/FCD Files

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

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

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

- Starting the Demo ................................................................. 19
- Demo Options ................................................................. 19
- Using the Demo .................................................................. 20
- Stopping the Demo .............................................................. 21

4.1 Starting the Demo

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

4.2 Demo Options

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

**Table 1. Command Line Switches**

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

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

- Establishing and Terminating a Call
- Keyboard Commands

4.3.1 Establishing and Terminating a Call

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

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

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

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

4.3.2 Keyboard Commands

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

---

Table 1. Command Line Switches (Continued)

<table>
<thead>
<tr>
<th>Switch</th>
<th>Action</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l&lt;n,...&gt;</td>
<td>Printouts will be printed into channel log files. If ‘all’ follows the –l, log files will be created for all available channels. If a list of channels in the following format: C1-C2, C3-C4, C5 follows the –l, log files are created for the channel ranges or specific channels specified in the list. If the “–l” option is not used, prints go to the stdout, for the first 2 channels only (to keep from overloading the CPU, and more convenient for viewing printouts).</td>
<td>Disabled</td>
</tr>
<tr>
<td>-m&lt;n,...&gt;</td>
<td>Enables printing channel specific information to the monitor, in addition to printing the log file. A maximum of 2 channels may be printed.</td>
<td>Disabled</td>
</tr>
<tr>
<td>-n&lt;ch&gt;</td>
<td>Sets the number of server channels</td>
<td>The lesser of Voice Devices or IP devices</td>
</tr>
</tbody>
</table>
### Running the Demo

#### 4.4 Stopping the Demo

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>d&lt;n&gt; or D&lt;n&gt;</td>
<td>Change debug level during runtime</td>
</tr>
<tr>
<td>e or E</td>
<td>Establish a call</td>
</tr>
<tr>
<td>m or M</td>
<td>Print log files for up to 2 channels to the screen</td>
</tr>
<tr>
<td>q or Q or CTRL+C</td>
<td>Terminate the application</td>
</tr>
<tr>
<td>t or T</td>
<td>Terminate a call</td>
</tr>
</tbody>
</table>
5. Demo Details

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

- Files Used by the Demo ................................................................. 23
- Programming Model Classes ...................................................... 25
- Threads .................................................................................. 31
- Initialization .......................................................................... 32
- Event Handling ...................................................................... 33

5.1 Files Used by the Demo

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

- Demo Source Code Files
- Utility Files
- PDL Files

5.1.1 Demo Source Code Files

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

- For Windows: C:\Program Files\dialogic\demo\ipdemo\multicastserver

<table>
<thead>
<tr>
<th>Directory</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>multicastserver</td>
<td>call.cpp</td>
<td>Implements the operations of the Call class</td>
</tr>
<tr>
<td>multicastserver</td>
<td>call.h</td>
<td>Function prototype for call.cpp</td>
</tr>
<tr>
<td>multicastserver</td>
<td>configuration.cpp</td>
<td>Implements the operations of the Configuration class</td>
</tr>
<tr>
<td>multicastserver</td>
<td>configuration.h</td>
<td>Function prototype for configuration.cpp</td>
</tr>
<tr>
<td>multicastserver</td>
<td>incfile.h</td>
<td>Function prototype for Global Call and R4 functions</td>
</tr>
<tr>
<td>multicastserver</td>
<td>ipmediaboard.cpp</td>
<td>Implements the operations of the IPMediaBoard class</td>
</tr>
<tr>
<td>multicastserver</td>
<td>ipmeadiaboard.h</td>
<td>Function prototype for ipmediaboard.cpp</td>
</tr>
<tr>
<td>multicastserver</td>
<td>ipmediadevice.cpp</td>
<td>Implements the operations of the IPMediaDevice class</td>
</tr>
<tr>
<td>multicastserver</td>
<td>ipmediadevice.h</td>
<td>Function prototype for ipmediadevice.cpp</td>
</tr>
<tr>
<td>multicastserver</td>
<td>main.cpp</td>
<td>Contains the main function and the Wait for Key</td>
</tr>
<tr>
<td>multicastserver</td>
<td>main.h</td>
<td>Function prototype for main.cpp</td>
</tr>
</tbody>
</table>
Table 3. Source Files Used by the IP Multicast Server (IPML) Demo (Continued)

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

5.1.2 Utility Files

The utility files listed in Table 4 are located in:

- For Windows: C:\Program Files\dialogic\demo\ipdemo\utilcpp\

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>utilcpp</td>
<td>utilcpp.ver</td>
<td>Utility library version information</td>
</tr>
<tr>
<td>utilcpp</td>
<td>log.cpp</td>
<td>Debugging functions</td>
</tr>
<tr>
<td>utilcpp</td>
<td>log.h</td>
<td>Function prototype for libdbg.c</td>
</tr>
<tr>
<td>utilcpp (Windows only)</td>
<td>utilcpp.dsw</td>
<td>Utility library Visual C++ workspace</td>
</tr>
<tr>
<td>utilcpp (Windows only)</td>
<td>utilcpp.dsp</td>
<td>Utility library Visual C++ project file</td>
</tr>
<tr>
<td>utilcpp/release (Windows only)</td>
<td>utilcpp.lib</td>
<td>Compiled Utility library</td>
</tr>
</tbody>
</table>
5.1.3 PDL Files

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>File Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdl_win</td>
<td>iptransport.cpp</td>
<td>PDL IP transport functions</td>
</tr>
<tr>
<td>pdl_win</td>
<td>iptransport.h</td>
<td>Function prototype for iptransport.cpp</td>
</tr>
<tr>
<td>pdl_win</td>
<td>pdl.c</td>
<td>Platform dependency functions</td>
</tr>
<tr>
<td>pdl_win</td>
<td>pdl.h</td>
<td>Function prototype for pdl.c</td>
</tr>
<tr>
<td>pdl_win</td>
<td>pdl.ver</td>
<td>PDL version information</td>
</tr>
<tr>
<td>pdl_win</td>
<td>pdl_win.dsp</td>
<td>PDL Visual C project file</td>
</tr>
<tr>
<td>pdl_win</td>
<td>pdl_win.dsw</td>
<td>PDL Visual C workspace</td>
</tr>
<tr>
<td>pdl_win\release</td>
<td>pdl_win.lib</td>
<td>Compiled PDL library</td>
</tr>
</tbody>
</table>

5.2 Programming Model Classes

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

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

5.2.1 Class Diagram

The following class diagram describes the relationship among the classes.
The main role of the Call class is to control all the resources related to a call. It contains all the resources related to a call and reflects the intersection of the call resource status.

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

Table 6. Call Class Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Access Privilege</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_pIPMediaDevice</td>
<td>public</td>
<td>IPMediaDevice*</td>
<td>The IPMedia device of the channel</td>
</tr>
<tr>
<td>m_pLog</td>
<td>public</td>
<td>Log*</td>
<td>A log instance</td>
</tr>
<tr>
<td>m_pConfiguration</td>
<td>public</td>
<td>Configuration*</td>
<td>Pointer to the configuration instance of the ResourceManager</td>
</tr>
<tr>
<td>m_pVoiceDevice</td>
<td>public</td>
<td>VoiceDevice*</td>
<td>The voice device of the channel</td>
</tr>
<tr>
<td>m_channelId</td>
<td>private</td>
<td>unsigned int</td>
<td>The channel identifier</td>
</tr>
<tr>
<td>m_currentState</td>
<td>private</td>
<td>E_StateMachine</td>
<td>The current state of the call</td>
</tr>
</tbody>
</table>
5.2.3 Configuration Class

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

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

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

5.2.4 IPMediaBoard Class

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

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Access Privilege</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_currentState</td>
<td>private</td>
<td>E_StateMachine</td>
<td>The current state of the channel</td>
</tr>
<tr>
<td>m_localMedialInfo</td>
<td>private</td>
<td>IPM_MEDIA_INFO</td>
<td>The local media information</td>
</tr>
<tr>
<td>m_mediaInfo</td>
<td>private</td>
<td>IPM_MEDIA_INFO</td>
<td>Used in the startMedia() function when calling the function ipm_SetRemoteMedialInfo(). The m_mediaInfo contains: the local RTP and RTCP information and the local coder information the remote RTP and RTCP information and the remote coder information</td>
</tr>
<tr>
<td>m_mediaStartedFlag</td>
<td>private</td>
<td>bool</td>
<td>Flag to indicate if the media has started</td>
</tr>
<tr>
<td>m_remoteMedialInfo</td>
<td>private</td>
<td>IPM_MEDIA_INFO</td>
<td>The remote GW media information</td>
</tr>
</tbody>
</table>

5.2.6 R4Device Class

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

The R4Device class attributes are described in Table 10.
5.2.7 R4LogicalBoard Class

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

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

Table 11. R4LogicalBoard Class Attributes

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

5.2.8 ResourceManager Class

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

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

The ResourceManager class attributes are described in Table 12.

Table 12. ResourceManager Class Attributes

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

5.2.9 VoiceBoard Class

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

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

Table 13. VoiceBoard Class Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Access Privilege</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_voiceDevices</td>
<td>public</td>
<td>VoiceDevice</td>
<td>Array of the voice devices found on the voice board</td>
</tr>
</tbody>
</table>
5.2.10 VoiceDevice Class

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

The VoiceDevice class attributes are described in Table 14.

Table 14. VoiceDevice Class Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Access Privilege</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_PlayerStopped</td>
<td>public</td>
<td>int</td>
<td>Flag that indicates if the player is stopped</td>
</tr>
<tr>
<td>prompt</td>
<td>public</td>
<td>DX_IOTT</td>
<td>Structure that identifies a source or destination for voice data. It is used with dx_play().</td>
</tr>
</tbody>
</table>

5.3 Threads

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

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

Figure 3. Thread Diagram
5.4 Initialization

Figure 4. IP Multicast Server (IPML) System Initialization

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

1. The application main( ) function sets up the callback handler, PDLsr_enbhdlr( ). The callback handler handles events that it receives from the SRL library. For more details see Section 5.5.3, “Handling SRL Events”, on page 33.

2. The application main( ) function then calls resourceManager.init( ), which does the following:
   a. Reads the command line options
   b. Reads and parse the configuration file and prints the configuration
   c. Gets the resources available in the system:
      – Gets the number of IP channels in the system
      – Gets the number of Voice channels in the system
      – Finds the minimum between the system channels and the user request
d. Looks for a free ipmedia device and returns a pointer to it

e. Opens the ipmedia device and if the open succeeds returns a pointer to it

f. Looks for a free voice device and returns a pointer to it

g. Opens the voice device and if the open success returns a pointer to it

h. Initializes the devices on the channel

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

5.5 Event Handling

This section contains the following topics:

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

5.5.1 Event Mechanism

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

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

5.5.2 Handling Keyboard Input Events

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

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

5.5.3 Handling SRL Events

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

1. Get the event device handle, by calling `PDLsr_getevtdv()`
2. Get the channel number related to the event, from the global array (HandleToChannel[ ])
3. Update the METAEVENT structure by calling `gc_GetMetaEvent()`
4. Get the event type, by calling `PDLsr_getevtttype()`
This chapter discusses the IP Multicast Server (IPML) state machines. It contains the following topics:

- Call State Machine ................................................................. 35
- IPMediaDevice State Machine .................................................. 37

6.1 Call State Machine

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

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

6.1.1 Call State Machine Description

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

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

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

The following state diagram describes the call states for the call class.
6.1.2 Call::callNull State

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

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

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

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

6.1.4 Call::callProceeding State

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

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

6.1.5 Call::callStopped State

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

6.2 IPMediaDevice State Machine

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

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

6.2.1 IPMediaDevice State Machine Description

The following state diagram describes the states for the IPMediaDevice class.
6.2.2 IPMediaDevice::mediaNull State

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

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

6.2.3 IPMediaDevice::mediaStarted State

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

If the application receives IPMEDIAEV_DROPCALL, it calls stopMedia() from the IPMediaDevice and the state transitions to mediaStopped.
6.2.4 IPMediaDevice::mediaCall State

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

6.2.5 IPMediaDevice::mediaStopped State

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

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

Codec: see COder/DECoder

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

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

DTMF: Dual-Tone Multi-Frequency

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

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

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

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

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

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

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

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

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

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

IFP: Internet Facsimile Protocol

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

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

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

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

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

ITU: See International Telecommunications Union.

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

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

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

PSTN: see Public Switched Telephone Network

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

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

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

RTCP: Real Time Control Protocol

RTP: Real Time Protocol

SIP: Session Initiation Protocol: an Internet standard specified by the Internet Engineering Task Force (IETF) in RFC 2543. SIP is used to initiate, manage, and terminate interactive sessions between one or more users on the Internet.
**T1:** A digital transmission link with a capacity of 1.544 Mbps used in North America. Typically channeled into 24 digital subscriber level zeros (DS0s), each capable of carrying a single voice conversation or data stream. T1 uses two pairs of twisted pair wires.

**TCP:** see Transmission Control Protocol

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

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

**UDP:** see User Datagram Protocol

**UDPTL:** Facsimile UDP Transport Layer protocol

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

**VAD:** Voice Activity Detection
Index

Symbols
{while(1)} 33

A
ATDV_SUBDEVS( ) 29

C
Call class 26
Call state machine 35
Call::callNull State 36
Call::callProceeding State 37
Call::callStarted State 37
Call::callStopped State 37
callback_hdlr( ) 33
class diagram 25
classes
  Call class 26
  Configuration class 27
  IPMediaBoard class 27
  IPMediaDevice class 28
  R4Device class 28
  R4LogicalDevice class 29
  ResourceManager class 29
  VoiceBoard class 30
  VoiceDevice class 31
compiling and linking 17
Configuration class 27
configuration files, editing 15

D
demo options 19
demo source code files 23
dt_Open( ) 29
dx_Open( ) 29
dx_play( ) 31

E
editing configuration files 15
establishing a call 20
event handling 33

F
files used by the demo 23

G
gc_GetMetaEvent( ) 33
gc_OpenEx() 28

H
handling keyboard input events 33
handling SRL events 33
hardware requirements 13

I
init() 33
initialization 32
ipm_SetRemoteMediaInfo( ) 28
IPMediaBoard class 27
IPMediaDevice class 28
IPMediaDevice state machine 37
IPMediaDevice::mediaCall State 39
IPMediaDevice::mediaNull State 38
IPMediaDevice::mediaStarted State 38
IPMediaDevice::mediaStopped State 39

K
keyboard commands 20
keyboard events, handling 33

L
listen() 36

M
m_mediaStartedFlag 38, 39
main() 32, 33
METAEVENT 33
PCD/FCD files, selecting 17
PDl files 25
PDlsr_enbdlr() 32, 33
PDlsr_gelevtdev() 33
PDlsr_gelevttype() 33
play() 37
preparing to run the demo 15
processEvent() 36, 37
processEvent(IPMEDIAEV_DROPCALL) 37
programming model classes 25

R4Device class 28
R4LogicalBoard class 29
requirements, hardware 13
ResourceManager class 29
resourceManager.init() 32
running the demo 19

selecting PCD/FCD files 17
software requirements 13
source code files 23
sr_enbhdlr() 31
SRL events, handling 33
starting the demo 19
startMedia() 28, 38
state machines
  Call state machine 35
  IPMediaDevice state machine 37
stopMedia() 38, 39
stopping the demo 21
stopPlay() 37
system requirements 13

terminating a call 20
threads 31

using the demo 20
utility files 24

VoiceBoard class 30
VoiceDevice class 31
waitForKey() 33