IP Mail (Global Call) Demo Guide

for Linux and Windows Operating Systems

05-1663-002

Publication Date: September, 2002

Document Number: 05-1663-002

Intel Converged Communications, Inc.
1515 Route 10
Parsippany NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support/

For **Products and Services Information**, visit the Intel Communications Systems Products website at: http://www.intel.com/network/csp/

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

Table of Contents

List of Tables

List of Figures

1. About This Information

The following topics provide information about this guide:

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

1.1. Purpose

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

1.2. Intended Audience

This information is intended for:

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

1.3. How to Use This Publication

The information in this guide is organized as follows:

- **Demo Description** introduces you to the demo and its features.
- **System Requirements** outlines the hardware and software required to run the demo.
- **Preparing to Run the Demo** describes the preparations required before running the demo.
- **Running the Demo** describes how to run the demo.
- **Demo Details** provides details on how the demo works.
- **Application Flow** describes the call control state machine.

- **Using the Voice Mail** describes the voice mail operation and state machine.

1.4. Related Information

See the following for more information:

- Release Guide for your system release – provides information about the system release, system requirements, software and hardware features, supported hardware, and release documentation
- Release Update for your system release (available on the Technical Support Web site only) – describes compatibility issues, restrictions and limitations, known problems, and late-breaking updates or corrections to the release documentation
- *http://developer.intel.com/design/telecom/support/* – Technical Support Web site which contains developer support information, downloads, release documentation, technical notes, application notes, a user discussion forum, and more

2. Demo Description

This chapter discusses the following topics:

- Demo Features
- What Does the IP Mail Demo Do?
- How Does the IP Mail Work?

2.1. Demo Features

The IP Mail demo illustrates how to build a simple Internet telephony voice mail application using the Global Call API.

The IP Mail demo allows the user to:

- Play pre-recorded announcements depending on system state
- Record new messages to a mailbox
- Listen to waiting messages

NOTE: The IP Mail demo does not function as a gateway. Therefore, it can answer calls only from the IP network. Gateway functionality can be added by connecting a gateway to interface with the PSTN.

The IP Mail demo is a cross-OS demo, running under the Linux* and Windows* operating system environments. Most of the differences in the environments are handled directly by the API and are transparent to the user. Other differences, due to inherent differences in the operating systems, are handled by the Platform Dependency Library (PDL). For more information, refer to the *PDL API Library Reference*.

2.2. What Does the IP Mail Demo Do?

The IP Mail demo implements a simple Internet telephony voice mail application. It receives a call from the IP network, answers the call and plays a series of voice menus. The demo recognizes the DTMF tones sent in response to the menu prompts, and acts in accordance with the specific tone received.

2.2.1. What the IP Mail Demo Does Not Do

- The demo does not allow setting the number of mailboxes on the fly. The number of mailboxes that are supported by the application is defined in the file *maildefs.h* (MAX_NUM_OF_MAILBOXS). The default is set to 200. Each mailbox can save only one message.
- The following features are not supported by this demo application:
    - UII message
    - NonStdCmd message
    - NonStdParm data
    - Q.931Facility message

2.3. How Does the IP Mail Demo Work?

The application answers the incoming call from the IP. When the H.323 connection has been established, the application uses the DM3 Player component to play the voice mail menu. The DM3 Signal Buffer sub-component is used to recognize the digits that were pressed by the user at the remote site. According to the digit pressed, the application performs the appropriate operation:

- Record new messages in a mailbox
- Listen to waiting messages

3. System Requirements

This chapter discusses the following topics:

- Hardware Requirements
- Software Requirements

3.1. Hardware Requirements

To run the IP Mail demo, you need:

- One of the following:
  - Intel® NetStructure™ DM/IP Series T1(E1)_NIC board
  - Intel® NetStructure™ IPT Series board

  NOTE: When running the demo with an Intel® NetStructure™ IPT Series board, an additional Intel® NetStructure™ board is required for playing/recording files for the demo.
- IP network cables

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

3.2. Software Requirements

To run the IP Mail demo, you need the Intel® Dialogic System Software for Linux or Windows. For a list of operating system requirements and supported compilers, see the Release Guide for your system release.

4. Preparing to Run the Demo

This chapter describes the preparations you must make before running the IP Mail demo. Topics discussed are:

- Connecting to External Equipment
- Editing Configuration Files
- Downloading the Firmware

4.1. Connecting to External Equipment

Use an Internet telephone or NetMeeting to connect directly via the IP, or connect to a gateway to connect via the PSTN network. See Figure 1.

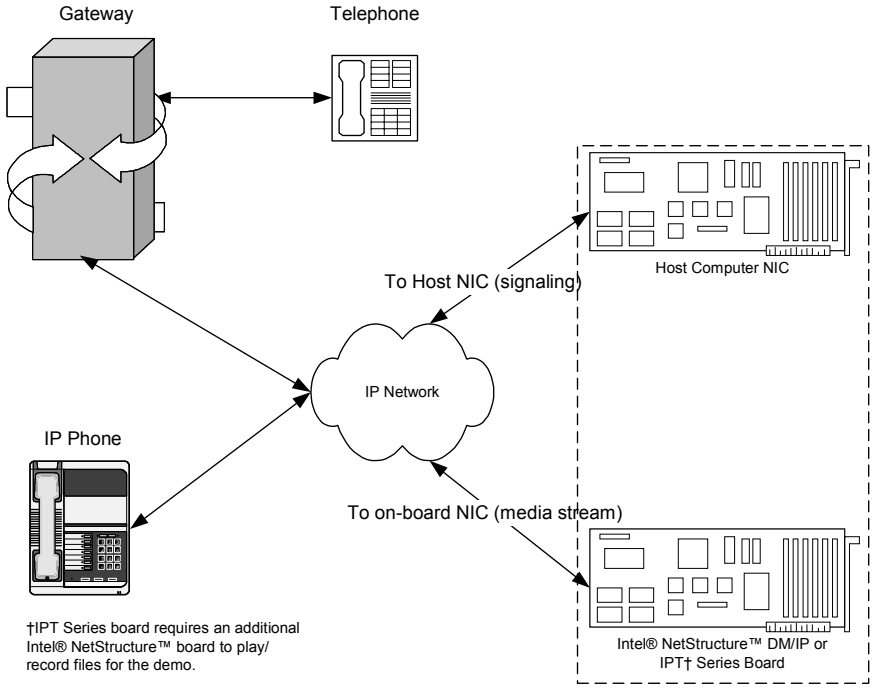

Figure 1. Connecting to External Equipment

4.2. Editing Configuration Files

This section shows a sample configuration file used with the IP Mail demo. Refer to the comments in the sample file for an explanation of the file contents.

Before running the IP Mail demo, modify the *iptmail_r4.cfg* file to reflect your system environment. Use a text editor and open the file from:

- **Windows:** *Dialogic\samples\gc_demos\IPDemos\iptmail_r4\release*
- **Linux:** */usr/dialogic/demos/ipt/gc_demos/iptmail_r4*

Sample Configuration File

Below is an example of the *iptmail_r4.cfg* file. Update the information about the coder that should be used when originating a call toward the IP network, according to your configuration.

```
##########################################################################################
# IP Protocol :
#    The IP Protocol used for  opening the IP Line devices, values: H323, SIP, both
#
# DTMFmode
#    possible options:
#         OutOfBand, inband, rfc2833
#
# Capability possibilities:
#    g711Alaw
#    g711Mulaw
#    gsm
#    gsmEFR
#    g723_5_3k
#    g723_6_3k
#    g729a
#    g729ab
#
#    Note: if you want to run the demo with coder g729 use:
#    g729a for running with VAD disable
#    and 729ab for running with VAD enable
#
# Caution:
#     If capability is g711Alaw /Mulaw  ==> FramesPerPkt = 10,20,30.
#                         G711 frame per packet defines the packet size in milliseconds
#     If capability is g723_5_3k / 6_3k ==> FramesPerPkt = 1, 2, 3 .
#                                           FrameSize isn't needed, default= 30ms.
#     If capability is gsm              ==> FramesPerPkt = 1, 2, 3 .
#                                           FrameSize isn't needed, default= 20ms.
#     If capability is gsmEFR           ==> FramesPerPkt = 1, 2, 3 .
#                                           FrameSize isn't needed, default= 20ms.
#     If capability is g729a            ==> FramesPerPkt = 3, 4 .
#                                           FrameSize isn't needed, default= 10ms.
#                                           VAD disable, the VAD parameter is ignored
#     If capability is g729ab            ==>FramesPerPkt = 3, 4 .
```

```
#                                          FrameSize isn't needed, default= 10ms.
#                                          VAD enable, the VAD parameter is ignored
#
##########################################################################################

ipProtocolName = H323
Channel = 1-120
{
   DTMFmode = OutOfBand

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

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

# MediaAlarmResetAlarmState        = 0

}
```

4.3. Downloading the Firmware - Windows

This section describes how to download the firmware on Windows systems. It describes the parameters that must be set before download.

4.3.1. Intel® NetStructure DM/IP Series Boards

Follow the directions for installing and configuring the DM/IP board in the Software Installation Guide for your system release and the Configuration Guide

for your board. The following parameters should be set in the Intel® Dialogic Configuration Manager (DCM) for the IP Mail demo to work properly:

- PCD/FCD Files

  Identify a PCD and corresponding FCD file that matches your configuration. Make sure to select a file that starts with **ipvs_evr**, in order to support the voice resources needed for the voice mail application. The PCD files supplied with the release are described in the Configuration Guide for your board.

- IP Address

  1. Select the Network tab.
  2. Enter the IP address of the NIC on the DM/IP board in the IPAddress field.
  3. Enter the Gateway IP address in the Gateway IPAddress field. This address should match the segment in the IPAddress field, ending with 250 (xxx.xxx.xxx.250).
  4. Click on the Start Service button to start the download.

4.3.2. Intel® NetStructure IPT Series Boards

At the time of writing this document, there are no special instructions for downloading the firmware. Refer to the Software Installation Guide for your system release and the Configuration Guide for your board for details on installing and downloading the firmware.

4.4. Downloading the Firmware - Linux

This section refers to both Intel® NetStructure IPT and DM/IP Series boards. Follow the directions in the Software Installation Guide for your system release and the Configuration Guide for your board for details on installing and downloading the firmware.

After completing the instructions in the Software Installation Guide and Configuration Guide:

1. Go to */usr/dialogic/bin*.
2. Run *dlstart* to download the firmware.

5. Running the Demo

This chapter discusses the following topics:

- Starting the Demo
- Demo Options
- Using the Demo
- Keyboard Commands

5.1. Starting the Demo

5.1.1. Windows

Select Run from the Start Menu. The demo executable file can be found in *C:\Program Files\Dialogic\samples\gc_demos\IPDemos\iptmail_r4\release\iptmail_r4.exe*. Click OK to run the IP Mail demo using the default settings.

5.1.2. Linux

The demo executable file can be found in */usr/dialogic/demos/ipt/gc_demos/iptmail_r4*.

5.2. Demo Options

To specify certain options at run-time, launch the demo from the command line, using any of the switches listed in *Table 1. IP Mail Command Line Switches*:

Table 1. IP Mail Command Line Switches

Switch	Action	Default
-n	Sets the number of channels	The lesser of voice devices or IP devices
-c <*filename*>	Configuration file name	iptmail_r4.cfg

Switch	Action	Default
-d	Sets debug level: 0 = Fatal - used when there is a non-recoverable error 1 = Error - same as fatal; used when there is a non-recoverable error 2 = Warning - used when some problem or failure has occurred that doesn't affect the channel's normal action 3 = Trace - used at the start of any function, or upon the successful completion of a function 4 = Info - prints data related to a specific action	0
-l	Printouts will be printed into channel log files in the demo directory. -lall = log files will be created for all available channels -l<channel list> = log files will be created only for the channel ranges or channels specified in the list **Note:** If the "-l" option is not used, all printouts will be to the stdout for channels 1 and 2 only.	Disabled
-q	Enables Quality of Service feature	Disabled
-h or ?	Prints the command syntax to the screen	Off

Each channel is initialized and the log for channels 1 and 2 is displayed in a separate window. See *Appendix A* for the log file from one channel of a typical session receiving a call from the IP.

5.3. Using the Demo

5.3.1. Using the Voice Mail

The IP Mail demo allows the caller to interact with a series of voice menus, using the telephone keypad to enter an option. Basic operations are playing a pre-recorded message and recording a new message. Each menu prompts the caller to select an action by pushing a key.

Main Menu [Main_Menu]
1 - Send Message
2 - Listen to Message
* - Quit

Send Message Prompt [Send_Message]
Enter Mailbox Number - between 101 - 299
* - Quit

Start Record Prompt [Start_Record]
2 - Start/Stop Record (at end Stop Record Prompt is played)
* - Quit

Stop Record Prompt [Stop_Record]
2 - Discard Message and re-record message to same mailbox
3 - Confirm Message (and Return to Main Menu) [Save_Confirm]
4 - Replay Message (and replay Stop Record Prompt)
* - Quit

Listen to Message Prompt [Listen_Menu]
- Enter Mailbox Number - between 101 - 299 (Recorded message is played)
* - Quit

Stop Listening Prompt [Stop_Listen]
2 - Discard Message and quit (and Return to Main Menu)
* - Quit

5.4. Keyboard Commands

While the demo is running, the administrator can send the following keyboard commands to the demo:

- q , ctrl+c - end application
- d - set debug level
- c - print channel information

6. Demo Details

This chapter discusses the following topics:

- Files Used by the Demo
- Programming Model
- Demo-Related Initialization
- IP Link-Related Initialization
- Voice-Related Initialization
- Data Structures
- Event Mechanism

6.1. Files Used by the Demo

This section lists and describes the following kinds of files that are used by the demo:

- Demo Source Files
- Utility Files
- PDL Files

6.1.1. Demo Source Files

In Windows, the following files are located in *Dialogic\samples\gc_demos\IPDemos\iptmail_r4*.

In Linux, the following files are located in */usr/dialogic/demos/ipt/gc_demos/iptmail_r4*.

Table 2. IP Mail Source Files

Filename	Description	OS
appdefs.h	Application definitions	Both
appinit.c	Application initialization	Both
appinit.h	Header file for application initialization	Both

Filename	Description	OS
appmain.c	The main application file: contains the thread initialization, route ports, etc.	Both
appmain.h	Header file for main application	Both
apppars.c	Read configuration file functions	Both
apppars.h	Header file for reading configuration file functions	Both
appstat.c	Application state machine functions	Both
appstat.h	Header file for the application state machine functions	Both
appstrc.h	Definitions of application structures	Both
appvars.h	Definition of application variables	Both
gateip.c	Initialize IP, get IP available channels, handle IP events	Both
gateip.h	Header file for the IP functions	Both
maildefs.h	Mailbox definitions	Both
mailutil.c	Mailbox utility functions	Both
mailutil.h	Header file for mailbox utility functions	Both
mediaalarms.c	QoS functions	Both
mediaalarms.h	Header file for the QoS functions	Both
voice.c	Initialize voice resources, get voice resources available channels, handle voice resources events	Both
voice.h	Header file for voice resources functions	Both
iptmail_r4.cfg	IP Mail demo configuration file	Linux
\release\iptmail_r4.cfg	IP Mail demo configuration file	Windows

Filename	Description	OS
iptmail_r4	IP Mail demo executable file	Linux
\release\iptmail_r4.exe	IP Mail demo executable file	Windows
stoplisten.vox	Voice file	Both
errorinput.vox	Voice file	Both
listenmenu.vox	Voice file	Both
mainmenu.vox	Voice file	Both
saveconfirm.vox	Voice file	Both
sendmsg.vox	Voice file	Both
startrec.vox	Voice file	Both
stoprec.vox	Voice file	Both
thankyou.vox	Voice file	Both
unavmenu.vox	Voice file	Both

NOTE: If you move the files for the IP Mail demo, be sure to move the *.vox* files as well, otherwise the IP Mail demo will not work.

6.1.2. Utility Files

In Windows, the following files are located in *Dialogic\samples\gc_demos\IPDemos\Util*.

In Linux, the following files are located in */usr/dialogic/demos/ipt/gc_demos/util*.

Table 3. IP Mail Utility Files

Filename	Description	OS
libdbg.c	Debugging functions	Both
libdbg.h	Function prototype for libdbg.c	Both

Filename	Description	OS
libdefs.h	#DEFINE inclusions	Both
libutil.a	Compiled Utility library	Both
makefile.util	Compilation file	Both
util.dsp	Utility library Visual C project file	Windows
util.dsw	Utility library Visual C workspace	Windows
util.ver	Utility library version information	Both
\release\util.lib	Compiled Utility library	Windows

6.1.3. PDL Files

In Windows, the following files are located in *Dialogic\samples\gc_demos\IPDemos\Pdl_nt.*

In Linux ,the following files are located in */usr/dialogic/demos/ipt/gc_demos/pdl_linux*.

Table 4. IP Mail PDL Files

Filename	Description	OS
libpdl.a	Compiled PDL library	Linux
makefile.pdl	Compilation file	Linux
pdl.c	Platform Dependency Library functions	Both
pdl.h	Function prototype for pdl.c	Both
pdl.ver	PDL version information	Both
pdl_nt.dsp	PDL Visual C project file	Windows
pdl_nt.dsw	PDL Visual C workspace	Windows
\release\pdl_nt.lib	Compiled PDL library	Windows

6.2. Programming Model

The IP Mail demo operates with two threads as shown in Figure 2. The threads are created as follows:

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

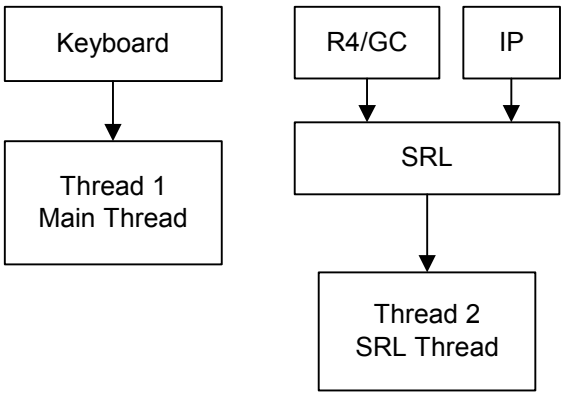

Figure 2. Thread Diagram

6.3. Demo-Related Initialization (Appinit.c)

The application **main( )** function calls **MailInitialization( )**, which does the following:

1. Gets any arguments from the command line.
2. Calls **IPTResetSession( )** to reset the demo data structures and initialize all channels' states to INIT.
3. Reads information from the configuration file (*iptmail_r4.cfg* or other CFG file determined by the user) and updates the ConfigFileParm in the AppSession data structure.

4. Calls **gc_Start( ):** starts all configured, call control libraries.
5. Sets up the call-back handler, **PDLsr_enbhdlr( ).**
6. Calls **getVoiceChannels( )** which checks the number of available voice devices:
   - Gets the number of voice boards, by calling **PDLsr_getboardcnt( )**.
   - For each board that was found:
     - Opens the board, by calling **dx_open( )**.
     - Finds number of channels per board, by calling **ATDV_SUBDEVS( )**.
     - Calculates the logical board and channel and saves them into AppSession.VoiceParams.
     - Closes the board, by calling **dx_close( )**.
7. Calls **getIPChannels( )** which checks the number of available IP channels:
   - Gets the number of voice boards, by calling **PDLsr_getboardcnt( )**.
   - For each board that was found:
     - Opens the board, by calling **gc_open( )**.
     - Finds number of IP channels per board, by calling **ATDV_SUBDEVS( )**.
     - Calculates the logical board and channel and saves them into AppSession.ipParams.
     - Closes the board, by calling **gc_close( )**.
8. Calls **getGateChannels( )** to find the demo MAX available channels (the smaller of available IP or Voice Devices and the number of channels specified with the –n command line option, if used).

   NOTE: The **printf( )** function must be used when printing an error/ trace/warning during the above initialization operations, because some actions are not related to a specific channel and log files are not yet opened.
9. Opens the channels' log files if –l command-line option was used.
10. Prints information from the configuration file.
11. Calls the **IPInit( )** function. See *Section 6.4. IP Link-Related* Initialization for a description of the **IPInit( )** function.

12. Calls the **VoiceInit( )** function. See *Section 6.5. Voice-Related Initialization* for a description of the **VoiceInit( )** function..
13. Calls the **InitMailBoxs( )** function, which initializes the MAILBOX structure.
14. The application **main( )** function calls **waitForKey( )**, to receive keyboard input.

6.4. IP Link-Related Initialization

The IP Link initialization procedure described in this section is defined in the **IPInit( )** function in the *gateip.c* file, which performs the following for all channels:

1. Calls **gc_OpenEx( )** in asynchronous mode. This opens all IP devices, and returns LineDevH.
2. Saves the channel number in the global array (HandleToChannel[ ]) according to the LineDevH handle.

6.5. Voice-Related Initialization

Call **VoiceInit( )**, located in the *voice.c* file, which does the following for all channels:

1. Calls **dx_open( )** to get the VoiceH (voice device handle).
2. Saves the channel number in the global array (HandleToChannel[ ]) according to the VoiceH handle.
3. Calls **dx_getxmitslot(VoiceH).** Returns the Xmitslot (the TDM bus time slot connected to transmit of voice channel) and saves it in the AppSession.VoiceParams structure.

6.6. Data Structures

This section describes the main data structures:

- AppSession
- IPPARAMS

- VOICEPARAMS
- MAILBOX
- CHANInfo
- CallParameters
- QoSElements

AppSession

```
typedef struct {
     unsigned short  sessionState;      /* Session state */
     appStateFxn     stateFxn;          /* Next state machine function to be called */
     unsigned int    sessionNumber;     /* Index session number */
     allParameters   ConfigFileParm;    /* Information that received from */
                                        /* gc_Extension() and from cfg file */
     FILE            *LogFile;          /* channel log file */
     LPMAILBOX       lpMailBox;         /* Pointer to the MailBox that is used */
     VOICEPARAMS     VoiceParams;       /* The voice device parameters */
     IPPARAMS        ipParams;          /* The IP device parameters */
     int             mediaDevice;       /* the media device handle that received */
                                        /* after calling gc_GetResourceH() */
     char            enteredExtNum[MAX_EXT_SIZE]; /* the Ext. number entered by */
                                        /* the user*/
     int             IsRouted;         /* If Route done then set flag to TRUE */
                                        /* else FALSE */
     QoSAlarmCounter QoSAlarmCount;     /* Counter for QoS alarms */
   }AppSession;
```

The AppSession data structure (defined in *appstrc.h*) contains the following fields:

- sessionState
  - the state machine current state
- stateFxn
  - the current state machine function according to sessionState
- sessionNumber
  - the channel number
- ConfigFileParm
  - information from the CFG file
- *LogFile
  - pointer to a session log file
- lpMailBox
  - pointer to the MailBox that is used

- VoiceParams
  - VOICEPARAMS structure
- ipParams
  - IPPARAMS structure
- enteredExtNum
  - the extension number entered by the user
- IsRouted
  - indicates if the call is routed for voice mail
- QoSAlarmCount
  - counter for Quality of Service alarms

There is an AppSession structure for every channel. Each channel session contains a pointer to a MAILBOX structure.

IPPARAMS

```
typedef struct {
     LINEDEV   linedev;   /* line device handle to identify the IP physical */
                          /* device that carries the call */
     long      XmitSlot;  /* the transmit timeslot number assigned for the Voice */
                          /* Resources */
     int       logBoard;  /* The logical board number for the session */
     int       logChan;   /* The logical channel number for the session */
     CRN       crn;       /* The current call's CRN */
} IPPARAMS;
```

The IPPARAMS data structure (defined in *appstrc.h*) contains the following fields:

- linedev
  - the line device handle, to identify the physical device(s) that carries the call
- XmitSlot
  - the time slot number for the IP
- logBoard
  - the logical board number
- logChan
  - the logical channel number

- crn
  - the Call Reference Number, generated after getting IP Offered

VOICEPARAMS

```
typedef struct {
     int   VoiceDevH;     /* the voice device handle */
     long  XmitSlot; /*the  transmit timeslot number assigned for the Voice Resources */
     int   logBoard;      /* The logical board number for the session */
     int   logChan;       /* The logical channel number for the session */
     DV_DIGIT digp[MAX_DIGITS_NUM];  /* A structure for the digits when getting */
                                     /* TDX_GETDIG event */
     DX_IOTT  iott;       /* data structure contains parameters for the Input/Output */
                          /* Transfer Table. */
     int  fPlayerStopped; /* A flag which indicates the Player status */
     int  fRecorderStopped;  /* A flag which indicates the Recorder status */
} VOICEPARAMS;
```

The VOICEPARAMS data structure (defined in *appstrc.h*) contains the following fields:

- VoiceDevH
  - the voice device handle
- XmitSlot
  - the time slot number for the voice resources
- logBoard
  - the logical board number
- logChan
  - the logical channel number
- digp
  - a structure (DV_DIGIT) for the digits when getting TDX_GETDIG event
- iott
  - a structure (DX_IOTT) for the file handle, played or recorded file
- fPlayerStopped
  - a flag which indicates the Player status
- fRecorderStopped
  - a flag which indicates the Recorder status

MAILBOX

```
typedef struct _MAILBOX {
    char ExtNum[MAX_EXTNUM_LENGTH+1];  /* mail-box number */
    MB_STATUS    msgFileStatus;        /* file status: 0 for no message,1 for saved */
                                       /* message */
    int          isBusy;               /* Mailbox busy: 0 for ready to answer an */
                                       /* incoming call, 1 for currently answering */
                                       /*a call */
}MAILBOX, *LPMAILBOX;
```

The MAILBOX data structure (defined in *maildefs.h*) contains the following fields:

- ExtNum
  - an array that contains the digits of the mailbox number
- msgFileStatus
  - 0 = no message
  - 1 = one saved message
- isBusy
  - 1 = mailbox is currently answering a call
  - 0 = mailbox is ready to answer an incoming call

CHANInfo

```
typedef struct {
   unsigned int      MadeRoute;          /* Calls that were routed */
   unsigned int      MadeUnRoute;        /* Calls that were unrouted */
   unsigned int      callsOffered;       /* Calls that were offered */
   unsigned int      callsDropped;       /* Calls that were dropped - when
                                          /*GCEV_DROPCALL received */
   unsigned int      droppingCalls;      /* Dropping calls - when calling to
                                          /*gc_DropCall */
   unsigned int      callsNull;          /* Calls that were null */
   unsigned int      callsTaskFailed;    /* Calls that were failed */
   unsigned int      callsTdxError;      /* Calls that were with Tdx error */
   unsigned int      callsAnswered;      /* Calls that were answered */
   unsigned int      callsDisconnected;  /* Calls that were disconnected  - when
                                          /*GCEV_DISCONNECTED received */
   unsigned int      callsMsgPlayed;     /* Calls that message was played */
   unsigned int      callsMsgRecorded;   /* Calls that message was recorder */
   }CHANInfo;
```

The CHANInfo data structure (defined in *appstrc.h*) contains the following fields:

- MadeRoute
  - number of calls that were routed

- MadeUnRoute
  - number of calls that were unrouted
- callsOffered
  - number of calls that were offered
- callsDropped
  - number of calls that were dropped when GCEV_DROPCALL was received
- droppingCalls
  - number of calls that were dropped when calling to **gc_DropCall( )**
- callsNull
  - number of calls that were null
- callsTaskFailed
  - number of calls that were failed
- callsTdxError
  - number of calls that were with Tdx error
- callsAnswered
  - number of calls that were answered
- callsDisconnected
  - number of calls that were disconnected when GCEV_DISCONNECTED was received
- callsMsgPlayed
  - number of calls that the message was played
- callsMsgRecorded
  - number of calls that a message was recorded

CallParameters

```
typedef struct {
   char             display[IPT_MAX_STRING];    /* Display null-terminated string */
   char             phoneList[IPT_MAX_STRING];  /* Phone list null-terminated string */
   char             callerId[IPT_MAX_STRING];   /* Caller Id address */
   unsigned int     callDurationTime;           /* Duration time in seconds */
   char            localPhoneNumber[IPT_MAX_STRING]; /* Local phone Number - used*/
                                              /* for inbound call without phoneList */
   char             srcAddr[IPT_MAX_STRING];    /* My IP address */
   char             destAddr[IPT_MAX_STRING];   /* Destination IP address */
   IP_CAPABILITY    TxCoder[MAX_CODER_CAPABILITY]; /* Receive Tx capability coders */
```

```
IP_CAPABILITY     RxCoder[MAX_CODER_CAPABILITY]; /* Receive Rx capability coders */

short             maxTxCoders;                   /* Max Tx capability coders */
short             maxRxCoders;                   /* Max Rx capability coders */
int               DTMFMode;                      /* DTMF modes */
QoSElements       QoSInfo;                       /* The QoS structure */

IP_RTCPINFO       RTCPInfo;                      /* RTCP information structure */
char              ConferenceID[16];              /* For conference call info use */
int               ConferenceGoal;                /* For conference call info use */
char              IPT_UUI[IPT_MAX_STRING];       /* User to User Information */
char              NonStdData[IPT_MAX_STRING];    /* Non standard data used in Q.931*/
                                                  /* and H.245 messages */
char              NonStdObjID[IPT_MAX_STRING];   /* Non standard objID used in */
                                                  /* Q.931and H.245 messages */
IP_H221NONSTANDARD H221NonStd;                   /* Vendor H221 structure */
char              productID[IPT_MAX_STRING];     /* Vendor product string */
char              versionID[IPT_MAX_STRING];     /* Vendor version string */
}CallParameters;
```

The CallParameters data structure (defined in *appstrc.h*) contains the following fields:

- display
  - display null-terminated string
- phoneList
  - phone-list null-terminated string
- callerId
  - caller ID address
- callDurationTime
  - duration time in seconds
- localPhoneNumber
  - local phone number - used for inbound call without phoneList
- srcAddr
  - originator IP address
- destAddr
  - destination IP address
- TxCoder
  - receive Tx capability coders
- RxCoder
  - receive Rx capability coders

- maxTxCoders
  - maximum number of Tx capability coders
- maxRxCoders
  - maximum number of Rx capability coders
- DTMFMode
  - DTMF modes
- QoSInfo
  - the QoS structure
- RTCPInfo
  - RTCP information structure
- ConferenceID
  - for conference call info use
- ConferenceGoal
  - for conference call info use
- IPT_UUI
  - User-to-User Information
- NonStdData
  - non-standard data used in Q.931 and H.245 messages
- NonStdObjID
  - non standard objID used in Q.931 and H.245 messages
- H221NonStd
  - vendor H221 structure
- productID
  - vendor product string
- versionID
  - vendor version string

QoSElements

```
typedef struct {
    IPM_QOS_THRESHOLD_INFO    QoSParamInfo;   /* The IPM_QOS_THRESHOLD_INFO includes */
                                               /* an array of IPM_QOS_THRESHOLD_DATA */
                                               /* {each one includes 7 fields, one for */
```

```
                                        /* the QoS type  (lost packets, jitter, */
                                        /* DTMF discarded) and the others for */
                                        /* the QoS thresholds to be read from */
                                        /* the configuration file}, and a */
                                        /* counter for the number of the */
                                        /* entries at the array */
   unsigned short           ResetAlarmState;
}QoSElements;
```

The QoSElements data structure (defined in *appstrc.h*) contains the following fields:

- QoSParamInfo
  - array containing QoS type and QoS thresholds
- ResetAlarmState
  - reserved for future use

6.7. Event Mechanism

The IP Mail 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 **mailInitialization( )** function sets up the call-back handler, by calling **PDLsr_enbhdlr( )**.

6.7.1. Handling Keyboard Input Events

There is an endless loop **{while(1)}** in the **main( )** function in the *appmain.c* 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.

6.7.2. Handling SRL Events

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

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

6.7.3. Handling Application Exit Events

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

7. Application Flow

This chapter discusses the following topics:

- Using State Machines
- Inbound IP Call

7.1. Using State Machines

All channels are initialized to the INIT state.

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 next state machine function is updated according to the event received and the sessionState.

7.2. Inbound IP Call

The following state diagram describes the call states for an inbound call from the IP to the demo. Each state is represented by an ellipse containing the state name, the event(s) associated with that state, and the actions performed by the application. The states are connected with arrows indicating the valid state changes.

NOTE: Due to space restrictions, the state name in the diagram is presented in shortened format. Each state name should be prefixed by IPTMAIL_.

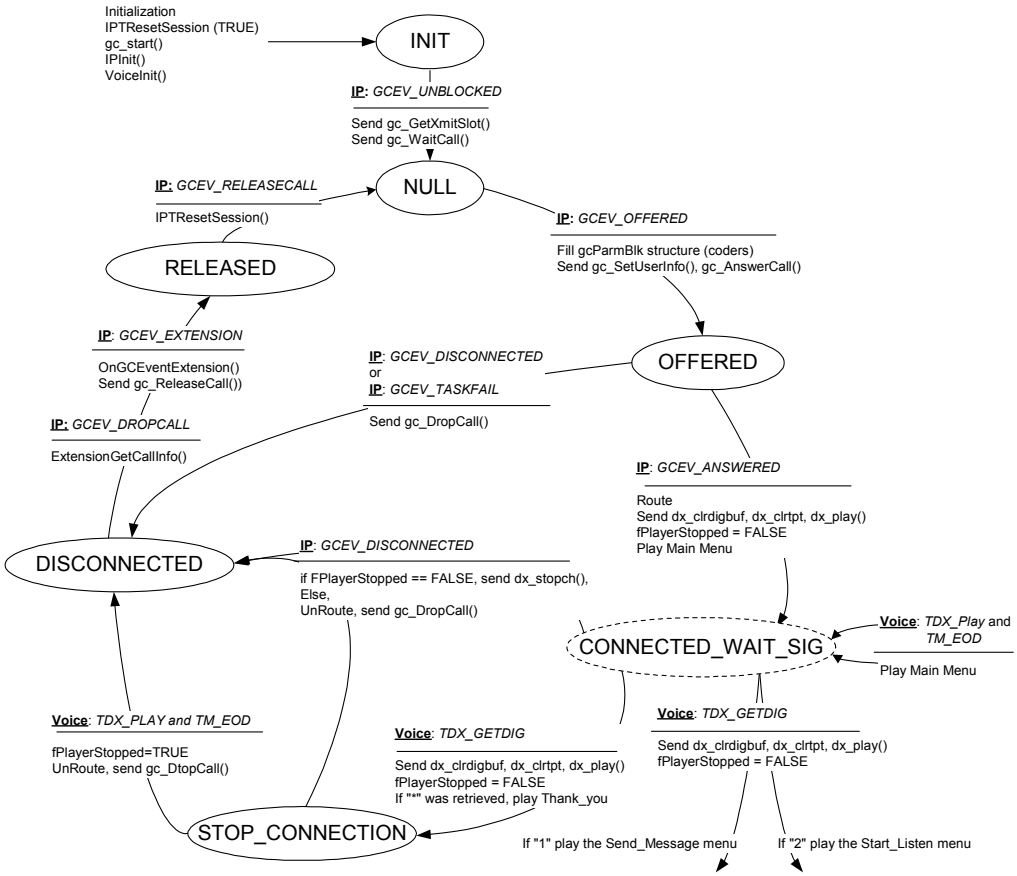

Figure 3. State Diagram — Inbound IP Call

7.2.1. Init State

The application waits for a GCEV_UNBLOCKED event in the IPTMAIL_INIT state. Upon receiving this event, the application calls **gc_GetXmitSlot(VoiceH)** to get the transmit time slot (Xmitslot) for the IP devices and saves it in the AppSession.ipParams structure.

The application then calls **gc_WaitCall( )** to set the conditions for processing an inbound call.

If the application receives GCEV_TASKFAIL, GCEV_BLOCKED, or GCEV_OPENEX_FAIL, it calls **endApplication( )** to gracefully shut down the application.

If the application receives GCEV_OPENEX, it does nothing so as to avoid causing an error.

7.2.2. Null State

The application waits for a call event in the IPTMAIL_NULL state.

When the application receives the event GCEV_OFFERED, it calls an internal function **ipAnswerCall( )** which fills the gcParmBlk structure with coder information, gets the protocol name, and sends **gc_SetUserInfo( )** and **gc_AnswerCall( )** to the firmware. The call state transitions to IPTMAIL_OFFERED.

If, for any reason, the call should fail, the application receives a GCEV_TASKFAIL event. The application calls **endApplication( )** to gracefully shut down the application.

7.2.3. Offered State

When the application receives a GCEV_ANSWERED event, it routes the IP and Voice devices to listen to each other, flushes the DTMF buffer, enables the DTMF tones used by the Main menu, and plays the Main menu. The call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

If, for any reason, the function should fail, the application receives a GCEV_TASKFAIL event. It calls the internal function **DisconnectCall** which, in turn, stops the I/O operation if in process, unroutes the call if needed, and calls **gc_DropCall( )**. The call state transitions to IPTMAIL_DISCONNECTED.

If, for any reason, the remote side should disconnect, the application receives a GCEV_DISCONNECTED event. It calls the internal function **DisconnectCall**

which, in turn, stops the I/O operation if in process, unroutes the call if needed, and calls **gc_DropCall( )**. The call state transitions to IPTMAIL_DISCONNECTED.

7.2.4. Connected_Wait_Sig State

The application waits for the user to make a selection from the Main menu:

- record a message ("1")
- listen to a message ("2")
- exit ("*")

When the application receives a TDX_PLAY event, it determines if the Player was stopped because it reached the end of data or a DTMF digit was detected. In the case of end of data (TM_EOD), the Player replays the menu. In the case of a digit (TM_MAXDTMF), the application sends **dx_getdig( )** to retrieve the digit.

The application waits in the IPTMAIL_CONNECTED_WAIT_SIG_STATE until it receives a TDX_GETDIG event.

If the retrieved digit is "1", the application plays the Send_Message menu. The call state transitions to IPTMAIL_CONNECTED_REC_AND_SEND.

If the retrieved digit is "2", the application plays the Start_Listen menu. The call state transitions to IPTMAIL_CONNECTED_START_LISTEN.

If the retrieved digit is "*", the application plays the Thank_You menu and the call state transitions to IPTMAIL_STOP_CONNECTION.

If no digit was retrieved (due to time-out or a non-enabled digit), the application replays the Main menu.

The application can also receive a GCEV_DISCONNECTED event. It calls an internal function **DisconnectCall( ),** which calls **dx_stopch( )** to prevent further playing, unroutes the clusters, and calls **gc_DropCall( )**. The call state transitions to IPTMAIL_DISCONNECTED.

Once the call state transitions to IPTMAIL_CONNECTED_WAIT_SIG, the demo application begins to operate in voice mail mode. See *Chapter 8. Using the Voice Mail* for a complete description of this part of the demo.

7.2.5. Disconnected State

The application waits for a GCEV_DROPCALL event. When it receives this event, it sends the **ExtensionGetCallInfo( )** function to get the call duration time and RTCP information. This information is returned in the GCEV_EXTENSION event. When it receives this event, the application calls **gc_ReleaseCall( )** in asynchronous mode. The call state transitions to IPTMAIL_RELEASED.

If the Player or Recorder hasn't yet been closed, the application waits for a TDX_PLAY or TDX_RECORD, closes the file, and calls **DisconnectCall( )** which in turn calls **gc_DropCall( )**.

7.2.6. Released State

The application waits for a GCEV_RELEASECALL event. When it receives this event, it sends **IPTResetSession( )** and the call state transitions to IPTMAIL_NULL.

If, for any reason, the call should fail, the application receives a GCEV_TASKFAIL event. The application calls **endApplication( )** to gracefully shut down the application.

7.2.7. Stop_Connection State

The application waits for a TDX_PLAY event. It requests the termination reason, closes the played file, unroutes the devices, and calls **gc_DropCall( )**. The call state transitions to IPTMAIL_DISCONNECTED.

The application may receive a GCEV_DISCONNECTED event. It checks to see if the Player has stopped. If PlayerStopped = FALSE, it sends **dx_stopch( )** and waits for TDX_PLAY and TM_EOD events. Upon receipt of these events, the application unroutes the clusters and calls **gc_DropCall( )**. The call state transitions to IPTMAIL_DISCONNECTED.

8. Using the Voice Mail

This chapter discusses the following topics:

- Basic Voice Mail Operation
- Recording a Message
- Listening to a Message
- Error Handling

8.1. Basic Voice Mail Operation

Each state function within the voice mail part of the application performs the same basic activities:

1. Waits for an IP and Voice Resource event.
2. Uses **dx_play( )** to play the appropriate menu. TDX_PLAY termination events will be generated to indicate completion.

   After getting the TDX_PLAY event, the application calls **ATDX_TERMMSK(chdev)** to check the termination reason. It may be either TM_EOD (end of data/file) or TM_MAXDTMF (user pressed digit/s as determined by the TPT – Termination Parameter Table).
   - If TM_EOD, then play the relevant menu again.
   - If TM_MAXDTMF, then collect the DTMF from the digit buffer.
3. Uses **dx_rec( )** to record a file. TDX_RECORD termination events will be generated to indicate completion.
4. Calls **dx_getdig( )** to **c**ollect the DTMF tone(s) from the digit buffer. TDX_GETDIG termination event will be generated to indicate completion.
5. Calls **dx_clrdigbuf( )** to clear all digits in the channel's firmware digit buffer.
6. Calls **dx_clrtpt( )** to clear the Table Parameter Termination (TPT) structure.

8.2. Recording a Message

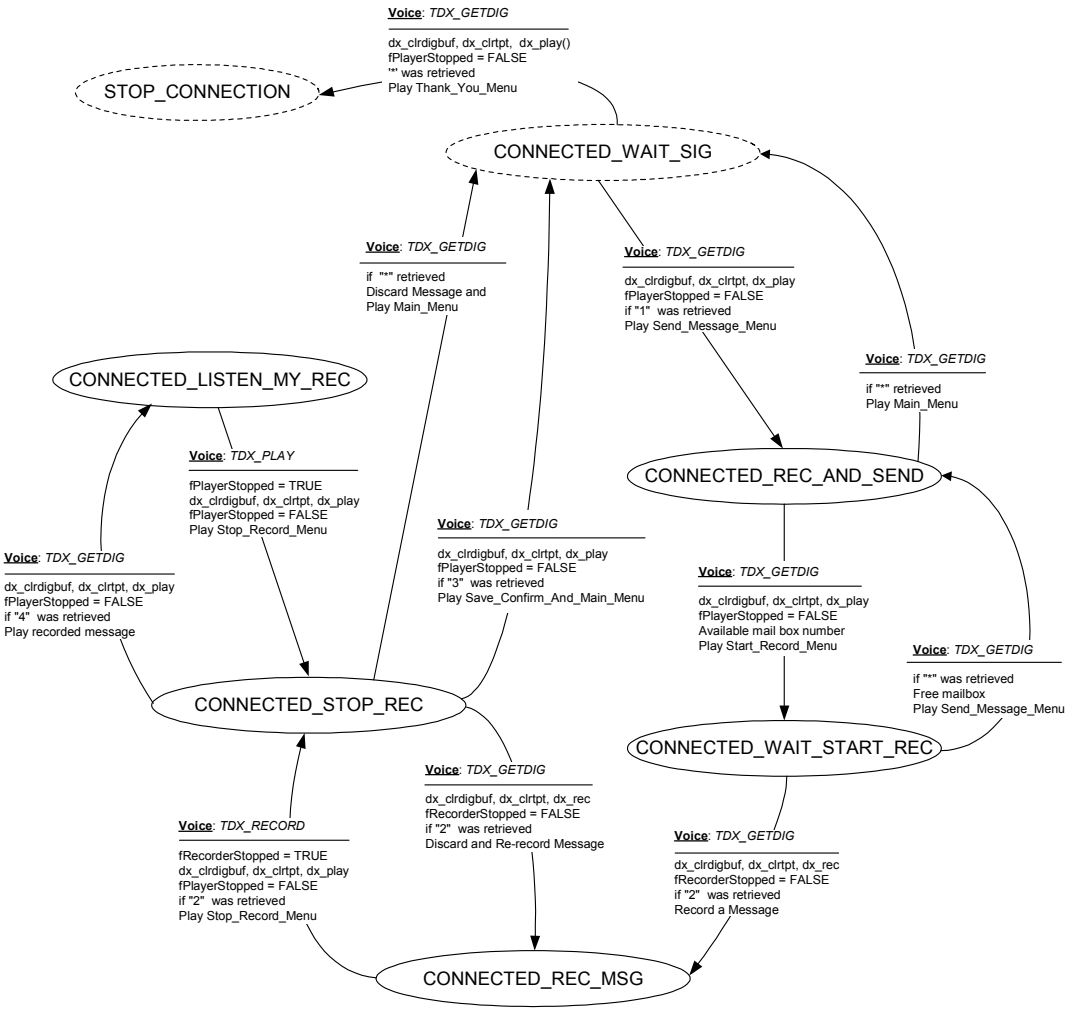

Figure 4. Record Message State Diagram

8.2.1. Connected_Rec_And_Send State

The application waits for a TDX_GETDIG event. The retrieved digit may be either an asterisk "*" or a 3-digit mailbox number. Any other digit or combination of digits generates an error.

If the digit is "*", the application replays the Main menu and the call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

If three digits are retrieved, the application checks if it is a legal mailbox number. If it is not, the application plays the ERROR_MENU and waits for another TDX_GETDIG event.

If the mailbox number is legal, the application checks if the requested mailbox is available. If it is unavailable, the application plays the UNAVAILABLE_TRY_AGAIN_MENU and waits for another TDX_GETDIG event.

If the mailbox number is available, the application plays the STARTREC_MENU and the call state transitions to IPTMAIL_CONNECTED_WAIT_START_REC.

If the application receives a TDX_PLAY event, it requests the termination reason. If it receives TM_EOD, the STOPREC_MENU is played. Otherwise it calls **dx_getdig( )** to collect the digit/s from the digit buffer. The application then waits for a TDX_GETDIG event. The call state remains in IPTMAIL_CONNECTED_REC_AND_SEND.

8.2.2. Connected_Wait_Start_Rec State

The application waits for a TDX_GETDIG event. The retrieved digit may be either an asterisk "*" or "2". Any other digit generates an error.

If the digit is "*", the application plays the SENDMSG_MENU. The call state transitions to IPTMAIL_CONNECTED_RECORD_AND_SEND.

If the digit is "2", the application does the following:

1. Calls **dx_clrtpt( )** to clear the TPT (Termination Parameter Table) structures.
2. Sets the DV_TPT structure to terminate on "2".

3. Sets up DX_IOTT (data structure that contains parameters for the Input/Output Transfer Table).
4. Calls **dx_clrdigbuf( )** to clear the digit from the buffer.
5. Calls **dx_rec( )** to begin recording on the channel.

The call state transitions to IPTMAIL_CONNECTED_RECORD_MSG.

8.2.3. Connected_Rec_Msg State

The application waits for a TDX_RECORD event. It gets the termination reason and calls **dx_fileclose( )** to close the recorded file.

The application plays the STOPREC_MENU and the call state transitions to IPTMAIL_CONNECTED_STOP_REC.

8.2.4. Connected_Stop_Rec State

The application waits for a TDX_GETDIG event. The retrieved digit may be either an asterisk "*", "2", "3", or "4". Any other digit generates an error.

If the returned digit is "2", the application discards the recorded message and plays the beep prompt to begin recording again. The call state transitions to IPTMAIL_CONNECTED_RECORD_MSG.

If the returned digit is "3", the application marks the mailbox as full and plays the SAVE_CONFIRM_AND_MAIN_MENU. The call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

If the returned digit is "4", the application plays the message back and the call state transitions to IPTMAIL_CONNECTED_LISTEN_MY_REC.

If the returned digit is "*", the application frees the mailbox and doesn't store the message. The application plays the MAIN_MENU and the call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

8.2.5. Connected_Listen_My_Rec State

The application waits for a TDX_PLAY event. When it receives the event, it closes the played file and plays the STOPREC_MENU. The call state transitions to IPTMAIL_CONNECTED_STOP_REC.

8.3. Listening to a Message

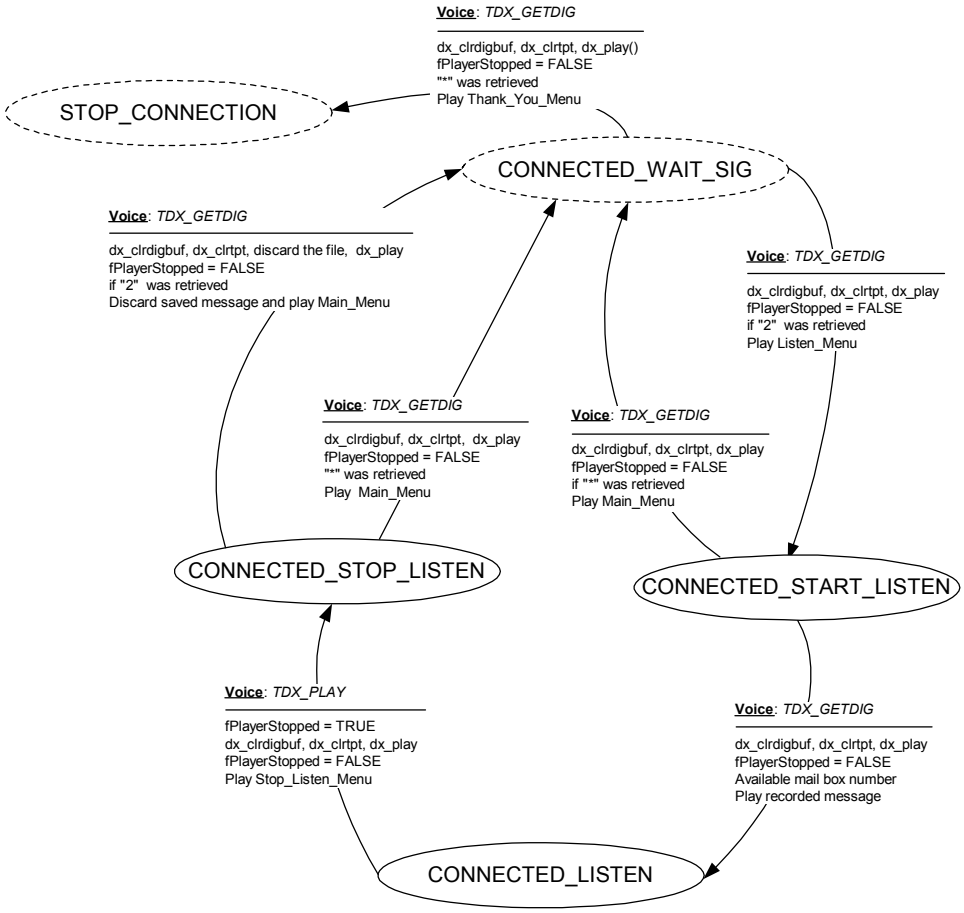

Figure 5. Listen to Message State Diagram

8.3.1. Connected_Start_Listen State

The application waits for a TDX_GETDIG event. The retrieved digit may be either an asterisk "*" or a 3-digit mailbox number. Any other digit or combination of digits generates an error.

If the digit is "*", the application replays the Main menu and the call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

If three digits are retrieved, the application checks if it is a legal mailbox number. If it is not, the application plays the ERROR_MENU and waits for another TDX_GETDIG event.

If the mailbox number is legal, the application checks if the requested mailbox is available. If it is unavailable, the application plays the UNAVAILABLE_TRY_AGAIN_MENU and waits for another TDX_GETDIG event.

If the mailbox number is available, the application plays the file and the call state transitions to IPTMAIL_CONNECTED_LISTEN.

8.3.2. Connected_Listen State

The application waits for a TDX_PLAY event. When it receives the event, it gets the termination reason, closes the played file, and plays the STOPLISTEN_MENU. The call state transitions to IPTMAIL_CONNECTED_STOP_LISTEN.

8.3.3. Connected_Stop_Listen State

The application waits for a TDX_GETDIG event. The retrieved digit may be either an asterisk "*" or "2". Any other digit or combination of digits generates an error.

If the digit is "*", the application preserves the message in the mailbox and plays the MAIN menu. The call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

If the digit is "2" the application discards the message, frees the mailbox, and plays the MAIN menu. The call state transitions to IPTMAIL_CONNECTED_WAIT_SIG.

8.4. Error Handling

The following diagrams illustrate how errors are handled by the IP Mail demo.

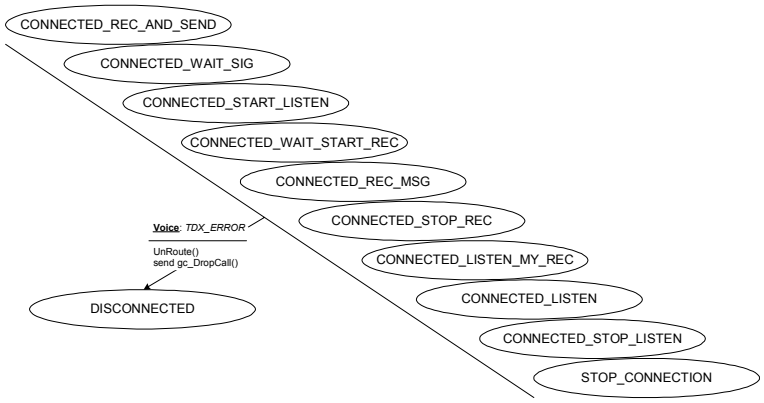

Figure 6. Error Handling—GCEV_DISCONNECTED

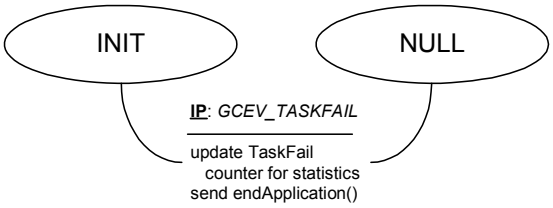

Figure 7. Error Handling—GCEV_TASKFAIL

8.4.1. Errors from IP

The error might occur in different phases:

- If the called function returns a <0 , first process the error by calling: **gc_ErrorValue( )** and **gc_ResultMsg( )**, and then end the call or exit the application.
- If the function was called successfully, but a GCEV_TASKFAIL event is received later, indicating that the function failed, the application will be terminated.

8.4.2. Errors from Voice Resources

- The main error which could be received while calling the R4 function toward the Voice Resources is represented by the TDX_ERROR event.
- The IP Mail demo drops the call after receiving this error and the state transitions to IPTMAIL_DISCONNECTED.

Appendix A

Log File

```
02/13/02  10:01:29
TRACE: File: gateip.c Line: 150
     Got TX timeslot(47) of the IP device on channel1

02/13/02  10:01:29
TRACE: File: voice.c Line: 200
     Got TX timeslot(96) of the voice device on channel1

02/13/02  10:01:29
TRACE: File: appstat.c Line: 62
In IPTMAIL_INIT on channel 1
      got Event GCEV_UNBLOCKED (0x833)

02/13/02  10:01:29
TRACE: File: appstat.c Line: 81
     gc_WaitCall was called on channel 1

02/13/02  10:01:32
TRACE: File: appstat.c Line: 139
In IPTMAIL_NULL  on channel 1
      got Event GCEV_OFFERED (0x824)

 02/13/02  10:01:32
TRACE: File: gateip.c Line: 311
     Answering Call on channel 1

 02/13/02  10:01:32
TRACE: File: appstat.c Line: 216
In IPTMAIL_OFFERED on channel 1
      got event GCEV_ANSWERED (0x802)

02/13/02  10:01:32
TRACE: File: appmain.c Line: 228
     Voice device succeed to listen to IP device timeslot, on channel 1.

 02/13/02  10:01:32
TRACE: File: appmain.c Line: 238
     IP device succeed to listen to Voice device timeslot, on channel 1.

02/13/02  10:01:32
TRACE: File: appstat.c Line: 251
     Route done on channel 1

 02/13/02  10:01:32
TRACE: File: voice.c Line: 288
     The VOX file: mainmenu.vox was opened on channel 1

 02/13/02  10:01:32
TRACE: File: appstat.c Line: 261
     Player started on channel 1

 02/13/02  10:01:38
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_PLAY (0x81)
```

```
 02/13/02  10:01:38
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

02/13/02  10:01:38
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

02/13/02  10:01:38
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:01:38
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:01:38
TRACE: File: appstat.c Line: 334
     Received digits: <1> on channel 1

 02/13/02  10:01:38
TRACE: File: voice.c Line: 288
     The VOX file: sendmsg.vox was opened on channel 1

 02/13/02  10:01:52
TRACE: File: appstat.c Line: 491
In IPMAIL_CONNECTED_RECORD_AND_SEND on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:01:52
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: End Of File

 02/13/02  10:01:52
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:01:52
TRACE: File: voice.c Line: 288
     The VOX file: sendmsg.vox was opened on channel 1

 02/13/02  10:02:03
TRACE: File: appstat.c Line: 491
In IPMAIL_CONNECTED_RECORD_AND_SEND on channel 1
      got event TDX_PLAY (0x81)

02/13/02  10:02:03
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:02:03
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:03
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1
```

```
 02/13/02  10:02:05
TRACE: File: appstat.c Line: 491
In IPMAIL_CONNECTED_RECORD_AND_SEND on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:02:05
TRACE: File: appstat.c Line: 511
     Received digits: <121> on channel 1

 02/13/02  10:02:05
TRACE: File: appstat.c Line: 576
     Available Mail_Box number on channel 1

 02/13/02  10:02:05
TRACE: File: voice.c Line: 288
     The VOX file: startrec.vox was opened on channel 1

 02/13/02  10:02:05
TRACE: File: appstat.c Line: 587
     Start Record Menu was played on channel 1

 02/13/02  10:02:17
TRACE: File: appstat.c Line: 685
In IPTMAIL_CONNECTED_WAIT_START_REC on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:02:17
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: End Of File

 02/13/02  10:02:17
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:17
TRACE: File: voice.c Line: 288
     The VOX file: startrec.vox was opened on channel 1

 02/13/02  10:02:17
TRACE: File: appstat.c Line: 685
In IPTMAIL_CONNECTED_WAIT_START_REC on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:02:17
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:02:17
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:17
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

02/13/02  10:02:17
TRACE: File: appstat.c Line: 685
In IPTMAIL_CONNECTED_WAIT_START_REC on channel 1
      got event TDX_GETDIG (0x83)
```

```
 02/13/02  10:02:17
TRACE: File: appstat.c Line: 706
     Received digits: <2> on channel 1

 02/13/02  10:02:17
TRACE: File: voice.c Line: 355
     The VOX file: MB121.vox was opened on channel 1

02/13/02  10:02:22
TRACE: File: appstat.c Line: 850
In IPTMAIL_CONNECTED_RECORD_MSG on channel 1
      got event TDX_RECORD (0x82)

 02/13/02  10:02:22
TRACE: File: appstat.c Line: 866
     Recorder stopped on channel 1.
     Reason: Specific digit received

 02/13/02  10:02:22
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:22
TRACE: File: voice.c Line: 288
     The VOX file: stoprec.vox was opened on channel 1

02/13/02  10:02:37
TRACE: File: appstat.c Line: 975
In IPTMAIL_CONNECTED_STOP_REC on channel 1
      got event TDX_PLAY (0x81)

02/13/02  10:02:37
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:02:37
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:37
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:02:37
TRACE: File: appstat.c Line: 975
In IPTMAIL_CONNECTED_STOP_REC on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:02:37
TRACE: File: appstat.c Line: 994
     Received digits: <4> on channel 1

 02/13/02  10:02:37
TRACE: File: voice.c Line: 288
     The VOX file: MB121.vox was opened on channel 1

 02/13/02  10:02:37
TRACE: File: appstat.c Line: 1053
     Play Recorded file on channel 1
```

```
 02/13/02  10:02:41
TRACE: File: appstat.c Line: 1184
In IPTMAIL_CONNECTED_LISTEN_MY_REC on channel 1
      got event TDX_PLAY (0x81)

02/13/02  10:02:41
TRACE: File: appstat.c Line: 1200
     Player stopped on channel 1.
     Reason: End Of File

 02/13/02  10:02:41
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:41
TRACE: File: voice.c Line: 288
     The VOX file: stoprec.vox was opened on channel 1

 02/13/02  10:02:41
TRACE: File: appstat.c Line: 1214
     Stop Record Menu was played on channel 1

 02/13/02  10:02:53
TRACE: File: appstat.c Line: 975
In IPTMAIL_CONNECTED_STOP_REC on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:02:53
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:02:53
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:02:53
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:02:53
TRACE: File: appstat.c Line: 975
In IPTMAIL_CONNECTED_STOP_REC on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:02:53
TRACE: File: appstat.c Line: 994
     Received digits: <3> on channel 1

 02/13/02  10:02:53
TRACE: File: voice.c Line: 288
     The VOX file: saveconfirm.vox was opened on channel 1

 02/13/02  10:02:53
TRACE: File: appstat.c Line: 1037
     Play Save_Confirm and MainMenu started on channel 1

 02/13/02  10:03:05
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_PLAY (0x81)
```

```
 02/13/02  10:03:05
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:03:05
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:05
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:03:05
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_GETDIG (0x83)

02/13/02  10:03:05
TRACE: File: appstat.c Line: 334
     Received digits: <2> on channel 1

 02/13/02  10:03:05
TRACE: File: voice.c Line: 288
     The VOX file: listenmenu.vox was opened on channel 1

 02/13/02  10:03:17
TRACE: File: appstat.c Line: 1305
In IPMAIL_CONNECTED_START_LISTEN on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:17
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:03:17
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:17
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:03:17
TRACE: File: appstat.c Line: 1305
In IPMAIL_CONNECTED_START_LISTEN on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:03:17
TRACE: File: appstat.c Line: 1325
     Received digits: <121> on channel 1

 02/13/02  10:03:17
TRACE: File: appstat.c Line: 1388
     Available Mail_Box number on channel 1

 02/13/02  10:03:17
TRACE: File: voice.c Line: 288
     The VOX file: MB121.vox was opened on channel 1

 02/13/02  10:03:17
```

```
TRACE: File: appstat.c Line: 1401
     Play Recorded file on channel 1

 02/13/02  10:03:21
TRACE: File: appstat.c Line: 1495
In IPTMAIL_CONNECTED_LISTEN on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:21
TRACE: File: appstat.c Line: 1511
     Player stopped on channel 1.
     Reason: End Of File

02/13/02  10:03:21
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:21
TRACE: File: voice.c Line: 288
     The VOX file: stoplisten.vox was opened on channel 1

 02/13/02  10:03:32
TRACE: File: appstat.c Line: 1620
In IPTMAIL_CONNECTED_STOP_LISTEN on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:32
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: End Of File

 02/13/02  10:03:32
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:32
TRACE: File: voice.c Line: 288
     The VOX file: stoplisten.vox was opened on channel 1

 02/13/02  10:03:33
TRACE: File: appstat.c Line: 1620
In IPTMAIL_CONNECTED_STOP_LISTEN on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:33
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:03:33
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:33
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:03:33
TRACE: File: appstat.c Line: 1620
In IPTMAIL_CONNECTED_STOP_LISTEN on channel 1
      got event TDX_GETDIG (0x83)
```

```
02/13/02  10:03:33
TRACE: File: appstat.c Line: 1639
     Received digits: <*> on channel 1

 02/13/02  10:03:33
TRACE: File: voice.c Line: 288
     The VOX file: mainmenu.vox was opened on channel 1

 02/13/02  10:03:33
TRACE: File: appstat.c Line: 1693
     Main Menu was played on channel 1

 02/13/02  10:03:38
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:38
TRACE: File: voice.c Line: 49
     Player stopped on channel 1.
     Reason: Receiving DTMF digits

 02/13/02  10:03:38
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

02/13/02  10:03:38
TRACE: File: voice.c Line: 445
     dx_getdig function sent successfully on channel 1

 02/13/02  10:03:38
TRACE: File: appstat.c Line: 313
IPTMAIL_CONNECTED_WAIT_SIG on channel 1
      got event TDX_GETDIG (0x83)

 02/13/02  10:03:38
TRACE: File: appstat.c Line: 334
     Received digits: <*> on channel 1

 02/13/02  10:03:38
TRACE: File: voice.c Line: 288
     The VOX file: thankyou.vox was opened on channel 1

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1796
In IPTMAIL_STOP_CONNECTION on channel 1
      got event TDX_PLAY (0x81)

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1812
     Player stopped on channel 1.
     Reason: End Of File

02/13/02  10:03:43
TRACE: File: voice.c Line: 508
     File was closed  on channel 1

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1819
     Player was stopped, and disconnecting the call on channel 1.

 02/13/02  10:03:43
```

```
TRACE: File: appmain.c Line: 417
     Making UnRoute on channel 1

 02/13/02  10:03:43
TRACE: File: appmain.c Line: 425
     Dropping IP call on channel 1.

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1901
In IPTMAIL_DISCONNECTED on channel 1
      got event GCEV_DROPCALL (0x805)

02/13/02  10:03:43
TRACE: File: gateip.c Line: 196
     gc_Extension(RTCPINFO,CALLDURATION) function was called on channel 1

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1901
In IPTMAIL_DISCONNECTED on channel 1
      got event GCEV_EXTENSION (0x868)

 02/13/02  10:03:43
INFO: File: gateip.c Line: 244
     Got extension data RTCPINFO:
     timestamp 1092704,
     tx_packets 3653,
     tx_octets 73060
     send_indication 1

02/13/02  10:03:43
INFO: File: gateip.c Line: 233
     Got extension data CALLDURATION: 130

 02/13/02  10:03:43
TRACE: File: appstat.c Line: 1968
     Releasing Call on channel 1.

 02/13/02  10:03:46
TRACE: File: appmain.c Line: 39
     Voice Device was closed on channel 1.

 02/13/02  10:03:46
TRACE: File: appmain.c Line: 48
     IP Line Device was closed on channel 1.
```

Index

G

H

I

K

L

M

N

P

Q

R

S

T

U

V