INTERNET CALL MANAGER
(ICM)

Syndicate Members

NC MOHIB RAZA
CSUO KASHIF ALAM
SGT USMAN MOHSIN
GC ZEESHAN KHALID

Directing Staff (DS)

LT COL AHSEN SAEED ZAIDI

A DISSERTATION

Submitted
To
Nation University of Sciences & Technology
In partial fulfillment of the requirements

For the degree of

Bachelors of Engineering (BE) Computer Software

Department of Computer Software Engineering
Military College of Signals

October 2001

ABSTRACT

The basic idea of this project is to use the same telephone line for Internet browsing and receiving telephone calls. Thus not missing any important phone call while on Internet. Our project comprises 2 servers and a module on a client side. Server 1 deals with the phone calls coming from the collee and server 2 deals with the client side that is connected on the Internet. These servers are implemented using JComm API, JMF API, JDBC, Motorola AT command set and Multithreading for the Internet clients.

ACKNOWLEDGEMENTS

First of all, Thanks to Allah Almighty for His continued help and guidance. We would also like to thank the following people for their support and guidance in completion of our project.

LT COL AHSEN SAEED ZAIDI	ISI
LT COL YOUNAS	EME
MAJ DR SHOAIB	EME/ET
CAPT ADIL	PASCOMS
MR ALI ASHAN	MCS
MR ARSHAD	INFONET
MR SHAHZAD	MARI GAS
MR ZAFAR	ALCATEL
MR OBAIDULLAH	SILICON
MR NAVEED	PTCL

We would also like to thank family members and colleagues for their continuous feedback and moral support.

TABLE OF CONTENTS

1. BACKGROUND

In order to check the feasibility of this project we carried out research work and came to know that this idea has never been implemented in Pakistan. Although there are some sites available in Canada and USA, there were very few servers, who were giving full real time communication, but majorities of them were based on either answering machine or voice messaging. But the problem with real time communication providers is that they wanted others to forward their calls to their number, which is highly infeasible for the local market of Pakistan, in terms of cost of long distance call charges.

During the research we went to Alcatel office and came to know that they are providing hardware equipment, which can be used for this purpose. But the cost of that server was round about 1 million. So it was unaffordable for general-purpose usage.

Then we started visiting different exchanges and ISP's for the feasibility of this project and finally decided to implement the idea using a PC having modem. So we started with a rough model and implemented the idea.

2. INTRODUCTION

When you're using your only phone line to surf the net, that line is completely tied up. Call Waiting won't help, and if you have voice mail, you won't know about any messages until you get off the line. That means people calling you get a busy signal or your voice mail, and you don't even know they called. Some of those calls could be important or even urgent - perhaps a call from a family member, client, customer, your boss - anyone! Did you know that approximately one third of all calls placed to households with Internet service receive a busy signal because someone is on the Internet at the time? That mean's you're missing a lot many calls because of busy line. We can use Internet Call Manager (ICM) to get rid of this problem.

ICM server provides a connection between a user and the telecom company while his telephone line is busy for a pp Internet connection and the user is connected on the Internet.

Thus connecting the incoming calls to the user through Internet using VOIP.

3. IP TELEPHONY

IP telephone is defined as any telephony application that can be enable across a packet-switched data network via the internet protocol. Packet-switched networks are not optimized for any one type of traffic, allowing intelligent end-user devices to encode and decode speech to make better use of available bandwidth. The voice compression algorithms now

used in IP-based telephony can deliver voice in a fraction of bandwidth used by circuit-switched cells. In addition, by treating voice another form of data and sending it over the same network as data, IP telephony is enabling new applications that use the best characteristics of voice communication and data processing. These applications can include PC-to-PC connection, PC to phone connections. Example applications include voice over the internet or internets, fax traffic (both real time and store-and-forward), unified message via web-enabled call centers, Internet call waiting, and much more.

Internet telephones have the potential to change the rules for long-distance telephone co. open the way to increased electronic commerce, and make corporate intranets more useful and productive. This is because Internet phones can improve communication while cutting travel and long distance costs. These benefits have already led more then two million people to use the Internet phones daily.

3.1 WHAT IS AN INTERNET TELEPHONE, AND WHAT CAN IT DO?

In a nutshell an internet telephone is a device that convert s voice into data for transmission over the internet, compresses that data for faster transmission, transmit the data in small “packets”, and then reassembles those packets, decompresses the data, and convert it back into voice at the other end. One advantage of IP telephony is that it dramatically improves efficiency of bandwidth use for real-time voice transmission, in many cases by a factor of 10 or more. Another advantage IP telephony has over the

PSTN is that it enables the creation of new class of service that includes applications such as web-enabled call centers, collaborative white boarding, and remote tele-working. This combination of human interaction and the power and efficiency of computers is opening up an entirely new world of communications. A final advantage of IP telephony is that it is additive to toady's communications networks. IP technology can be used in conjunction with the existing PSTN leased and dial-up lines, PBXs and other customer premise equipment (CPE), local area networks (LANs), and the internet connections. IP telephony applications can be implemented through dedicated servers, which in turn can be based on the state of art PC hardware and software platforms plus interface boards providing computer technology (CT), internet and IP telephony functions.

3.2 SYSTEM RESOURCE USAGE

The amount of system bandwidth used by the Internet telephony conversations varies directly and proportionally with the number of simultaneous conversations, and inversely with the amount of data compression used. Unfortunately, the trade off for additional compression is compromised by audio quality.

Bandwidth assumption with the Internet phones runs as little as 6720 bps per conversation, meaning that even 9600-baud modems can support Internet telephony. This level of consumption is efficient for standalone desktops using a dedicated telephone line, but for intranets that are local area networks (LAN) based and wide area networks (WAN) based, multiple conversations can quickly eat up bandwidth capacity. This problem is

execrated by the fact that Internet telephony software is designed to run in the background to allow multitasking, meaning that the same user can transmit other data to the network at the same time.

For example on a network with 100 users, 50 concurrent Internet phone conversations can eat up 350 kbps or more bandwidth, in addition to the current load on the network. Because many networks do not operate well with sustained loads over 30 to 50 % of capacity, this level of consumption can be significant. Indeed, it equates to almost 25% of a WAN TI connection.

3.3 AREAS OF APPLICATION

The main goal of VOIP is to piggyback voice and fax calls over an IP data network to save on long distance charges. A secondary goal is to incorporate IP voice and fax in to certain applications for enhanced services. These two goals are the primary focus in seven main VOIP market applications. Market Applications Corporate tool By-pass Toll free intra-company voice and fax between corporate locations. Fax over the Internet Tool free or reduced rate fax machines, fax between any two locations. PC phone to PC phone Tool free voice between two PC's on the Internet. IP based public phone services new public phone services, at reduced rates (especially internationally), where voice is sent over the Internet or over the new public IP networks. Voice is phone to phone, or phone to PC.Call center IP telephony, agent click new IP voice applications that allows a PC user on the internet to click on a phone icon in a catalog in the customer services homepage and talk to an agent via the PC as a phone. IP line Double A PC user at a home or in a hotel etc with just one connection to the internet

would subscribe to a new service that allows the single phone line to carry one or more phone calls in addition to the PC data. Premise IP telephony PC's in a building on an IP LAN would be able to make phones calls to ordinary phones in the same building or to make outside calls, using special VOIP equipment on the premise.

3.4 PRIVACY ISSUES

Most Internet phone software sends and receives audio directly between the users without going through a central server, although the server might track who is on line. The real time protocol (RTP) allows for the encryption of the multimedia stream between the conference members. This means that Internet phone calls are usually hard to trace or listen on in.

3.5 THE FUTURE OF INTERNET TELEPHONY

Both private and public organizations are working to connect conventional telephones with Internet telephones big industry giants such as NTT DoCoMo, AT&T, Cisco and Lucent Technologies have realized the importance of Internet Telephony in the future and are investing heavily in such applications. According to study by the International Data Corporation

(IDC), the current growth rate in the Internet Telephony industry is 150% and the market value is expected to reach §3.6 Billion by the year 2002.

3.6 OVERVIEW OF EXISTING VOICE OVER IP STANDARDS

When the IP telephony equipment manufacturers began to move their technologies from the laboratory into the real world, it became clear that the technical challenge of building a scalable network of end devices and gateways was greater than expected. The technologies needed to encode and transmit voice and fax traffic had been perfected, but the art of call control and address management for large corporate or service provider platforms still needed to evolve. As a result a number of protocols have been defined that allow IP telephony systems to inter-communicate. This section examines three common IP telephony protocols used in systems to date: H.323, MGCP, and SIP. These protocols lead an intertwined existence often being combined in many applications.

4. SOFTWARE ARCHITECTURE

4.1 PROJECT SPECIFICATIONS

SOFTWARE REQUIREMENTS

- SOFTWARE MODULE ON THE CLIENT SIDE.
- PHONE LISTENER SERVER FOR HANDLING TELEPHONE CALLS
- CLIENT LISTENER SERVER FOR MANAGING DATABASE.
- JMF
- JCOMM

HARDWARE REQUIREMENTS

- CALL FORWARDING ON INTERNET USER SIDE
- MULTIMEDIA SUPPORT
- FULL DUPLEX VOICE MODEM
- TELEPHONE LINE

4.2 CLASS HIERARCHY

4.2.1 CLIENT LISTENER SERVER SIDE

4.2.1.1 CLASS HIERARCHY FOR CLIENT LISTENER CLASSES

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container

- class java.awt.Window (implements javax.accessibility.Accessible)
  - class java.awt.Frame (implements java.awt.MenuContainer)
    - class **ClientListener**

- class **DBManager**

4.2.1.2 LIST OF ALL MEMBER FUNCTIONS

C

checkIP(String) - Method in class DBManager
This method is used check the duplicate IP address in the database.

checkTelNo(String) - Method in class DBManager
This method is used check the duplicate telephone number in the database.

ClientListener - class ClientListener.
This class is used to make the threads for the clients.

ClientListener() - Constructor for class ClientListener
Constructor only sets the GUI and Action Listeners.

con - Variable in class DBManager

D

DBManager - class DBManager.
This class is handling all database transactions.

DBManager() - Constructor for class DBManager
Constructor only loads the driver necessary for the data base connections and then establishes the connection with the database.

deleteTelTuple(String) - Method in class DBManager
This function is used to delete the Telephone number from the data base.

deleteTuple(String) - Method in class DBManager
This function is used to delete the IP address from the data base.

display(String) - Method in class ClientListener
This function is only used to display the Strings on the GUI.

I

insertRequest(String, String, String) - Method in class DBManager

This function is called for any request of log on from the Internet user.

insertSequence(String, String, String) - Method in class DBManager

This function is called for any request of log on from the Internet user.

M

main(String[]) - Static method in class ClientListener

This function is only used to start the application.

R

runServer() - Method in class ClientListener

It receives the requests of the clients and call the ServerThread class to handle it.

4.2.1.3 CLASS CLIENTLISTENER

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--ClientListener
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable

public class **ClientListener**
extends java.awt.Frame

This class is used to make the threads for the clients. Clients of this class *PhoneListener class* and *IUser class*

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

IUser, ServerThread, Serialized Form

Inner classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Inner classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Inner classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Inner classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR,

NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary	
ClientListener() Constructor only sets the GUI and Action Listeners.	

Method Summary	
void	**display**(java.lang.String s) This function is only used to display the Strings on the GUI.
static void	**main**(java.lang.String[] arg) This function is only used to start the application.
void	**runServer**() It recieves the requests of the clients and call the ServerThread class to handle it.

Methods inherited from class java.awt.Frame

addNotify, finalize, getAccessibleContext, getCursorType, getFrames, getIconImage, getMenuBar, getState, getTitle, isResizable, paramString, remove, removeNotify, setCursor, setIconImage, setMenuBar, setResizable, setState, setTitle

Methods inherited from class java.awt.Window
addWindowListener, applyResourceBundle, applyResourceBundle, dispose, getFocusOwner, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getOwnedWindows, getOwner, getToolkit, getWarningString, hide, isShowing, pack, postEvent, processEvent, processWindowEvent, removeWindowListener, setCursor, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, remove, removeAll, removeContainerListener, setFont, setLayout, update, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener,

removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Constructor Detail

ClientListener

public ClientListener()

Constructor only sets the GUI and Action Listeners.

Method Detail

runServer

public void runServer()

It receives the requests of the clients and call the ServerThread class to handle it. It's clients are PhoneListener class and IUser class.

See Also:

ServerThread, IUser, ModemManager

---

display

public void display(java.lang.String s)

This function is only used to display the Strings on the GUI.

Parameters:

s - string to be dispalyed

---

main

public static void main(java.lang.String[] arg)

This function is only used to start the application.

Parameters:

arg[] - array of string from the command line, not applicable here

4.2.1.4 CLASS DBMANAGER

java.lang.Object

|

+--DBManager

---

public class **DBManager**
extends java.lang.Object

This class is handling all database transactions.

Version:

1.0, 9-Oct-2001

Author:

kusm

See Also:

ServerThread

Field Summary	
java.sql.Connection	**con**

Constructor Summary
DBManager() Constructor only loads the driver necessary for the data base connections and then establishes the connection with the database.

Method Summary	
int	**checkIP**(java.lang.String checkIP) This method is used check the duplicate IP address in the database.
int	**checkTelNo**(java.lang.String checkTel) This method is used check the duplicate telephone number in the database.
int	**deleteTelTuple**(java.lang.String anyTel) This function is used to delete the Telephone number from the data base.
int	**deleteTuple**(java.lang.String anyIP) This function is used to delete the IP address from the data base.
int	**insertRequest**(java.lang.String name, java.lang.String telNo, java.lang.String ipAdd) This function is called for any request of log on from the Internet user.
int	**insertSequence**(java.lang.String name, java.lang.String telNo, java.lang.String ipAdd) This function is called for any request of log on from the Internet user.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

con

public java.sql.Connection **con**

Constructor Detail

DBManager

public **DBManager**()

Constructor only loads the driver necessary for the data base connections and then establishes the connection with the database.

Method Detail

insertSequence

public int **insertSequence**(java.lang.String name,
java.lang.String telNo,
java.lang.String ipAdd)

This function is called for any request of log on from the Internet user. First of all it is checked that this ip or tel no is present in the data base or not if yes then they are deleted and new information inserted otherwise information is inserted as it is.

Parameters:

name - name of the internet user

telNo - telephone number of the internet user

ipAdd - IP address of the internet user

Returns:

status of the request,1 -> successful, 0 -> not successful

See Also:

ServerThread, checkIP(java.lang.String), #CheckTelNo, deleteTuple(java.lang.String), deleteTelTuple(java.lang.String), insertRequest(java.lang.String, java.lang.String, java.lang.String)

---

insertRequest

public int **insertRequest**(java.lang.String name,

java.lang.String telNo,

java.lang.String ipAdd)

This function is called for any request of log on from the Internet user.

Parameters:

name - name of the internet user

telNo - telephone number of the internet user

ipAdd - IP address of the internet user

Returns:

status of the request,1 -> sucessful, 0 -> not successful

See Also:

insertSequence(java.lang.String, java.lang.String,

java.lang.String)

---

deleteTuple

public int deleteTuple(java.lang.String anyIP)

This function is used to delete the IP adress from the data base. It may be in the case that user wants to log off or somehow he was not able to log off and new request for log on is of the same ip then previous entry will be deleted from the data base using this function.

Parameters:

anyIP - ip address to be deleted

Returns:

result of the query, 1 -> success, 0 -> no success

See Also:

insertSequence(java.lang.String, java.lang.String,

java.lang.String), ServerThread

---

deleteTelTuple

public int deleteTelTuple(java.lang.String anyTel)

This function is used to delete the Telephone number from the data base. It may be in the case that user was not able to log off and new request for log on is of the same number, then previous entry will be deleted from the data base using this function.

Parameters:

anyTel - Telephone number to be deleted

Returns:
result of the query, 1 -> success, 0 -> no success

See Also:

insertSequence(java.lang.String, java.lang.String, java.lang.String)

---

checkTelNo

public int checkTelNo(java.lang.String checkTel)

This method is used check the duplicate telephone number in the database. It takes the telephone number to be checked. end return the result.

Parameters:

checkTel - telephone number to be checked

Returns:
return the result of the query 1 -> success, 0 -> no success

See Also:

insertSequence(java.lang.String, java.lang.String, java.lang.String)

---

checkIP

public int checkIP(java.lang.String checkIP)

This method is used check the duplicate IP address in the database. It takes the IP address to be checked. and return the result.

Parameters:

checkIP - IP address to be checked

Returns:
return the result of the query 1 -> success, 0 -> no success

See Also:

insertSequence(java.lang.String, java.lang.String, java.lang.String)

4.2.2 IUSER SIDE

4.2.2.1 CLASS HIERARCHY FOR IUSER CLASSES

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Window (implements javax.accessibility.Accessible)
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class **IUser**

4.2.2.2 LIST OF ALL MEMBER FUNCTIONS

C

connectCL() - Method in class IUser
This function is used for Logging on.

D

disConnectCL() - Method in class IUser
This function is used for Logging off.

display(String) - Method in class IUser
This function dispalys the data in the text area to the user.

I

IUser - class IUser.

This class builds the GUI for the user and does the remaining all tasks.

IUser() - Constructor for class IUser

This constructor is used to build the GUI.

M

main(String[]) - Static method in class IUser

This function is used to make the object of the class.

S

sendData(String) - Method in class IUser

This function flushes the data in the output stream.

W

waitForCall() - Method in class IUser

This function initiate the session with the server.

4.2.2.3 CLASS IUSER

```
java.lang.Object
 |
 +--java.awt.Component
     |
     +--java.awt.Container
         |
         +--java.awt.Window
             |
             +--java.awt.Frame
                 |
                 +--javax.swing.JFrame
                     |
                     +--IUser
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class **IUser**
extends javax.swing.JFrame
This class builds the GUI for thr user and does the remaining all tasks.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

IUManager, ServerThread, Serialized Form

Inner classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Inner classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Inner classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Inner classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Inner classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR,

E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary	
IUser() This constructor is used to build the GUI.	

Method Summary	
void	**connectCL**() This function is used for Logging on.
void	**disConnectCL**() This function is used for Logging off.
void	**display**(java.lang.String show)

	This function dispalys the data in the text area to the user.
static void	**main**(java.lang.String[] arg) This function is used to make the object of the class.
void	**sendData**(java.lang.String data) This function flushes the data in the output stream.
void	**waitForCall**() This function initiate the session with the server.

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isRootPaneCheckingEnabled, paramString, processKeyEvent, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getFrames, getIconImage, getMenuBar, getState, getTitle, isResizable, remove, removeNotify, setCursor, setIconImage, setMenuBar, setResizable, setState, setTitle

Methods inherited from class java.awt.Window
addWindowListener, applyResourceBundle, applyResourceBundle, dispose, getFocusOwner, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getOwnedWindows, getOwner, getToolkit,

getWarningString, hide, isShowing, pack, postEvent, processEvent, removeWindowListener, setCursor, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFont, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation,

getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Constructor Detail

IUser

public **IUser**()

This constructor is used to build the GUI. iIt also sets the actoin listeners for the LOG OFF and LOG ON butons.

Method Detail

sendData

public void **sendData**(java.lang.String data)

This function flushes the data in the output stream.

Parameters:

data - String to be sent

---

display

public void **display**(java.lang.String show)

This function displays the data in the text area to the user.

Parameters:

show - String to be shown

---

connectCL

public void **connectCL**()

This function is used for Logging on. It sends request to the *ServerThread* class for logging on.

See Also:

ServerThread

---

disConnectCL

public void **disConnectCL**()

This function is used for Logging off. It sends request to the *ServerThread* class for logging off.

See Also:

ServerThread

---

waitForCall

public void **waitForCall**()

This function initiate the session with the server. It creates the Server socket and listen for the response from the *PhoneListener*.Then it makes the object of *Test* class to initiate voice conversation.

See Also:

Test, ServerThread

---

main

public static void **main**(java.lang.String[] arg)

This function is used to make the object of the class.

Parameters:

arg[] - any command line argument,null in this case

4.2.3 MODEM MANAGER SERVER SIDE

4.2.3.1 CLASS HIERARCHY FOR MODEM MANAGER SERVER

- class java.lang.Object
  - class **CLManager**
  - class **IUManager**
  - class **ModemManager**

4.2.3.2 LIST OF ALL MEMBER FUNCTIONS

C

callCenter() - Method in class ModemManager

This function is the main function of the class which performs all the tasks.

CLManager - class CLManager.

This class is used for interaction with the Data Base.

CLManager() - Constructor for class CLManager

It opens the connection with the *ServerThread class* to get the IP of the user.

command(String) - Method in class ModemManager

It is used to send the commands to modem through stream.

G

getTelNoIP(String) - Method in class CLManager
It passes the telephone number to the data base to get the IP of the Internt user from there.

giveNumber() - Method in class ModemManager
It is used to extract the telephone no.

I

initializeModem(CommPortIdentifier, String) - Method in class ModemManager
This method is used to initialized the modem.

isUserLogon(String) - Method in class IUManager
This is the main function of the class.It checks whether the user is offline or online and tells him/her that there is a call for this phone number.

IUManager - class IUManager.
This class is used for interaction with the Internet User.

IUManager() - Constructor for class IUManager

L

lookIn(char) - Method in class ModemManager
It is used to extract the only the telephone no.

M

main(String[]) - Static method in class ModemManager
It is the main function to run the aplication.

ModemManager - class ModemManager.
This class is used to interact with the modem.

ModemManager() - Constructor for class ModemManager
This constructor makes the object of the classes which are needed during the execution.

N

numberBolo(char) - Method in class ModemManager
It is used to tell the dialed telephone no.

numberNikalo() - Method in class ModemManager
It is used to send *numberNikalo* numbers to say it aloud to user.

P

print(byte[]) - Method in class ModemManager
It is used to print responses of the modem.

R

response() - Method in class ModemManager
It is used to take the response from the modem through stream.

T

telNoExtractor() - Method in class ModemManager
It is used to take the telephone number from user.

V

validateNo() - Method in class ModemManager
It is used to validate the number which is entered by the user.

4.2.3.3 CLASS MODEMMANAGER

java.lang.Object

|

+--ModemManager

---

public class **ModemManager**
extends java.lang.Object

This class is used to interact with the modem. This class also uses objects of *IUManager, CLManager and MessagePlayer* to accomplish the task.

Version:
1.0, 9-Oct-2001

Author:
ksum

See Also:
IUManager, CLManager, MessagePlayer

Constructor Summary	
ModemManager() This constructor makes the object of the classes which are needed during the execution.	

Method Summary	
void	**callCenter**() This function is the main function of the class which performs all the tasks.
void	**command**(java.lang.String atComm) It is used to send the commands to modem through stream.
void	**giveNumber**() It is used to extract the telephone no.
void	**initializeModem**(CommPortIdentifier portId, java.lang.String COMport) This method is used to initialized the modem.
void	**lookIn**(char c) It is used to extract the only the telephone no.
static void	**main**(java.lang.String[] args) It is the main function to run the aplication.
void	**numberBolo**(char awaz) It is used to tell the dialed telephone no.
void	**numberNikalo**() It is used to send *numberNikalo* numbers to say it aloud to user.
void	**print**(byte[] received) It is used to print responses of the modem.
byte[]	**response**() It is used to take the response from the modem through

	stream.
void	**telNoExtractor**() It is used to take the telephone number from user.
void	**validateNo**() It is used to validate the number which is entered by the user.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ModemManager

public ModemManager()

This constructor makes the object of the classes which are needed during the execution. Secondly it also locates the ports available to find the modem.

Method Detail

initializeModem

public void initializeModem(CommPortIdentifier portId,
java.lang.String COMport)

This method is used to initialized the modem. It first creates the streams with the modem and then give it AT commands to bring in it online voice command mode to recieve the telephone calls.

Parameters:

portId - id of the required port

COMport - port at which modem is coonected

---

command

public void command(java.lang.String atComm)

It is used to send the commands to modem through stream.

Parameters:

atComm - AT command to be send

See Also:

#initializedModem

---

response

public byte[] response()

It is used to take the response from the modem through stream. It takes response in the form of Byte array.

Returns:

response from the modem

See Also:

command(java.lang.String)

---

callCenter

public void callCenter()

This function is the main function of the class which performs all the tasks. it recieves the telephone call, give responses to the user,call other objects to connect to database and internet user etc. Finally it makes the object of test class to start conversation.

See Also:

IUManager, CLManager, MEssagePLayer, Test,

numberNikalo(), telNoExtractor(), #telNoBolo, validateNo()

---

print

public void print(byte[] received)

It is used to print responses of the modem.

Parameters:

received - Bytes to be printed.

See Also:

response()

---

telNoExtractor

public void telNoExtractor()

It is used to take the telephone number from user.

See Also:

giveNumber()

---

lookIn

public void lookIn(char c)

It is used to extract the only the telephone no. from the response of user.

See Also:

telNoExtractor()

---

giveNumber

public void giveNumber()

It is used to extract the telephone no. from the response of it.

See Also:

lookIn(char)

---

numberNikalo

public void numberNikalo()

It is used to send *numberNikalo* numbers to say it aloud to user.

See Also:

numberBolo(char)

---

numberBolo

public void numberBolo(char awaz)

It is used to tell the dialed telephone no. to user.

See Also:

numberNikalo()

---

validateNo

public void validateNo()

It is used to validate the number which is entered by the user.

See Also:

callCenter()

---

main

public static void **main**(java.lang.String[] args)

It is the main function to run the aplication.

See Also:

ModemManager()

4.2.3.4 CLASS IUMANAGER

java.lang.Object

|

+--IUManager

---

public class **IUManager**
extends java.lang.Object

This class is used for interaction with the Internet User. This class checks whether IUser is online or offline.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

ModemManager, IUser, CLManager

Constructor Summary	
IUManager()	

Method Summary	
int	**isUserLogon**(java.lang.String IP) This is the main function of the class.It checks whether the user is

	offline or online and tells him/her that there is a call for this phone number.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

IUManager

public IUManager()

Method Detail

isUserLogon

public int isUserLogon(java.lang.String IP)

This is the main function of the class.It checks whether the user is offline or online and tells him/her that there is a call for this phone number. It takes the IP of the Internet user from th e ClManger nad user it for checking purposes.

Parameters:

IP - IP of the Internet user

See Also:

CLManager, ModemManager

4.2.3.5 CLASS CLMANAGER

java.lang.Object

|

+--CLManager

---

public class **CLManager**
extends java.lang.Object

This class is used for interaction with the Data Base. This class is used to send the telephone number to the database to get the ip adress of the respective number from there.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

ClientListener, IUManager, ModemManager

Constructor Summary
CLManager() It opens the connection with the *ServerThread class* to get the IP of the user.

Method Summary	
int	**getTelNoIP**(java.lang.String tel) It passes the telephone number to the data base to get the IP of the Internt user from there.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CLManager

public CLManager()

It opens the connection with the *ServerThread class* to get the IP of the user.

See Also:

ServerThread

Method Detail

getTelNoIP

public int **getTelNoIP**(java.lang.String tel)

It passes the telephone number to the data base to get the IP of the Internt user from there.

Parameters:

tel - telephone number of the called party.

See Also:

ServerThread

4.2.4 MESSAGE PLAYER SIDE

4.2.4.1 CLASS HIERARCHY FOR MESSAGE PLAYER CLASSES

- class java.lang.Object
  - class java.util.Dictionary
    - class java.util.Hashtable (implements java.lang.Cloneable, java.util.Map, java.io.Serializable)
      - class **SoundList**
  - class **MessagePlayer**
  - class **SoundLoader**

4.2.4.2 LIST OF ALL MEMBERS FUNCTIONS

G

getClip(String) - Method in class SoundList

This function gets the audio clip to be palyed.

M

MessagePlayer - class MessagePlayer.

This class is used to play the messages to the phone user.

MessagePlayer() - Constructor for class MessagePlayer

P

putClip(AudioClip, String) - Method in class SoundList
This function puts the audio clip to be palyed.

S

SoundList - class SoundList.
This class Loads and holds a bunch of audio files whose locations are specified.
SoundList(URL) - Constructor for class SoundList
This constructor constructs the url and inturn the list of the files to be loaded.
SoundLoader - class SoundLoader.
This class is used to load thefiles of the messages to be played for the phone user.
SoundLoader(SoundList, URL, String) - Constructor for class SoundLoader
This constructor does all the job of loading the sounds.
start(String, int) - Method in class MessagePlayer
This function is used to start the clip once loaded.
startLoading(String) - Method in class SoundList
This function start loading the files basing on the relative url by passing the information to *SOundLoader* class.
stop() - Method in class MessagePlayer
This function is used stop the clip.

4.2.4.3 CLASS MESSAGEPLAYER

java.lang.Object
|
+--**MessagePlayer**

---

public class **MessagePlayer**
extends java.lang.Object
This class is used to play the messages to the phone user. This class loads all the messages before hand and palys them for the user on demand.
Version:
1.0, 9-Oct-2001

Author:

ksum

See Also:

ModemManager, SoundList, SoundLoader

Constructor Summary	
MessagePlayer()	

Method Summary	
void	**start**(java.lang.String chosenFile, int time) This function is used to start the clip once loaded.
void	**stop**() This function is used stop the clip.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MessagePlayer

public **MessagePlayer**()

Method Detail

stop

public void **stop**()

This function is used stop the clip.

---

start

public void **start**(java.lang.String chosenFile,
int time)

This function is used to start the clip once loaded.

Parameters:

chosenFile - name of the file to be started

time - interval for which execution will not proceed so that message can be played

---

4.2.4.4 CLASS SOUNDLIST

java.lang.Object
 |
 +--java.util.Dictionary
     |
     +--java.util.Hashtable
         |
         +--**SoundList**

All Implemented Interfaces:

java.lang.Cloneable, java.util.Map, java.io.Serializable

---

public class **SoundList**
extends java.util.Hashtable
This class Loads and holds a bunch of audio files whose locations are specified. relative to a fixed base URL.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

MessagePlayer, SoundLoader, Serialized Form

Inner classes inherited from class java.util.Map
java.util.Map.Entry

Constructor Summary	
SoundList(URL baseURL) This constructor constructs the url and inturn the list of the files to be loaded.	

Method Summary	
AudioClip	**getClip**(java.lang.String relativeURL) This function gets the audio clip to be played.
void	**putClip**(AudioClip clip, java.lang.String relativeURL) This function puts the audio clip to be played.
void	**startLoading**(java.lang.String relativeURL) This function start loading the files basing on the relative url by passing the information to *SoundLoader* class.

Methods inherited from class java.util.Hashtable
clear, clone, contains, containsKey, containsValue, elements, entrySet, equals, get, hashCode, isEmpty, keys, keySet, put, putAll, rehash, remove, size, toString, values

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

SoundList

public **SoundList**(URL baseURL)

This constructor constructs the url and inturn the list of the files to be loaded. For this process it uses *SoundLoader class*

Parameters:

baseURL - base url of the file

See Also:

SoundLoader

Method Detail

startLoading

public void **startLoading**(java.lang.String relativeURL)

This function start loading the files basing on the relative url by passing the information to *SOundLoader* class.

Parameters:

relativeURL - relative url of the file

relativeURL - relative url of the file to be loaded

See Also:

SoundLoader

---

getClip

public AudioClip **getClip**(java.lang.String relativeURL)

This function gets the audio clip to be palyed.

Parameters:

relativeURL - relative url of the file

Returns:

audio clip of the relative URL

---

putClip

public void **putClip**(AudioClip clip,
java.lang.String relativeURL)

This function puts the audio clip to be palyed.

Parameters:

relativeURL - relative url of the file

clip - clip to be put

4.2.4.5 CLASS SOUNDLOADER

java.lang.Object
|
+--**SoundLoader**

---

public class **SoundLoader**
extends java.lang.Object

This class is used to load thefiles of the messages to be played for the phone user. This class helps in loading all the messages before hand and palys them for the user on demand.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

ModemManager, SoundList, MessagePlayer

Constructor Summary
SoundLoader(SoundList soundList, URL baseURL, java.lang.String relativeURL) This constructor does all the job of loading the sounds.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SoundLoader

public **SoundLoader**(SoundList soundList,
URL baseURL,
java.lang.String relativeURL)

This constructor does all the job of loading the sounds.

Parameters:

soundList - list of files to be loaded

baseURL - base URL of the file

relativeURL - relative URL of the file

4.2.5 VOICE TRANSMISSION SIDE

4.2.5.1 CLASS HIERARCHY FOR VOICE TRANSMISSION

- class **VTServer**
- class **VTClient**

- class java.lang.Object
  - class **Test**
- class **MyControllerListener**

4.2.5.2 LIST OF ALL MEMBER FUNCTIONS

C

close() - Method in class MyControllerListener
This function is used to close the palyer.
configure(int) - Method in class MyControllerListener
This function is used to configure the palyer.
controllerUpdate(ControllerEvent) - Method in class MyControllerListener
This function is used for event handling.
createManager(String, int, int, boolean, boolean) - Method in class VTClient
This function is used to cerate the session with the Server.

D

devices_actionPerformed(String) - Method in class VTServer
This function is used to cerate the session with the client.

M

MyControllerListener - class MyControllerListener.
This class is used for transition of the player from one state to other.

P

playToEndOfMedia(int) - Method in class MyControllerListener
This function is used to detect end of media stream for the palyer.
prefetch(int) - Method in class MyControllerListener
This function is used for prefetching.

R

realize(int) - Method in class MyControllerListener

This function is used to realize the palyer.

T

Test - class Test.
This class use makes the objects of *VTServer* and *VTClient*
Test(String) - Constructor for class Test
It takes the IP address of the client.

U

update(ReceiveStreamEvent) - Method in class VTClient

update(RemoteEvent) - Method in class VTClient
This function is used to handle RemoteEvent.
update(SendStreamEvent) - Method in class VTServer
This function is used to handle SendStreamEvent.

V

VTClient - class VTClient.
This class initiate the Audio session with the Server.
VTClient() - Constructor for class VTClient

VTServer - class VTServer.
This class initiate the Audio session with the client.
VTServer() - Constructor for class VTServer

4.2.5.3 CLASS TEST

java.lang.Object

|

+--Test

---

public class **Test**
extends java.lang.Object

This class use makes the objects of *VTServer* and *VTClient*

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

VTServer, VTClient

Constructor Summary	
Test(java.lang.String ip) It takes the IP address of the client.	

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Test

public **Test**(java.lang.String ip)

It takes the IP address of the client. It passes the *IP address* to the VTServer method and to VTClient method createManager.

See Also:

createManager

4.2.5.4 CLASS VTSERVER

VTServer

---

public class **VTServer**

This class initiate the Audio session with the client. It sends request to VTClient to initiate the session. This class implements SendStreamListener and RemoteListener interfaces.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

VTClient

Constructor Summary	
VTServer()	

Method Summary	
void	**devices_actionPerformed**(java.lang.String ip) This function is used to cerate the session with the client.
void	**update**(SendStreamEvent evt) This function is used to handle SendStreamEvent.

Constructor Detail

VTServer

public VTServer()

Method Detail

devices_actionPerformed

public void devices_actionPerformed(java.lang.String ip)

This function is used to cerate the session with the client. This is the main function of the class which performs all the task of of the class.

Parameters:

ip - IP address of the client with which session will be initialized.

---

update

public void update(SendStreamEvent evt)

This function is used to handle SendStreamEvent.

Parameters:

evt -

4.2.5.5 CLASS VTCLIENT

VTClient

---

public class **VTClient**

This class initiate the Audio session with the Server. It accepts request of VTServer to initiate the session. This class implements ReceiveStreamListener and RemoteListener interfaces.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

VTServer

Constructor Summary	
VTClient()	

Method Summary	
SessionManager	**createManager**(java.lang.String address, int dataport, int cport, boolean listener, boolean sendlistener) This function is used to cerate the session with the Server.
void	**update**(ReceiveStreamEvent event)

void	**update**(RemoteEvent revent) This function is used to handle RemoteEvent.

Constructor Detail

VTClient

public **VTClient**()

Method Detail

createManager

public SessionManager **createManager**(java.lang.String address,
int dataport,
int cport,
boolean listener,
boolean sendlistener)

This function is used to cerate the session with the Server. This is the main function of the class which performs all the task of of the class.

Parameters:

address - IP address of the client with which session will be initialized.

dataport - data port for the communicaton

cport - control port for the communication

listener -

sendlistener -

Returns:
SessionManager

---

update

public void **update**(ReceiveStreamEvent event)

---

update

public void update(RemoteEvent revent)

This function is used to handle RemoteEvent.

Parameters:

revent -

4.2.5.6 CLASS MYCONTROLLERLISTENER

MyControllerListener

---

public class **MyControllerListener**

This class is used for transition of the player from one state to other.

Version:

1.0, 9-Oct-2001

Author:

ksum

See Also:

VServer

Method Summary	
void	**close**() This function is used to close the palyer.
boolean	**configure**(int timeOutMillis) This function is used to configure the palyer.
void	**controllerUpdate**(ControllerEvent ce) This function is used for event handling.
boolean	**playToEndOfMedia**(int timeOutMillis) This function is used to detect end of media stream for the palyer.

boolean	**prefetch**(int timeOutMillis) This function is used for prefetching.
boolean	**realize**(int timeOutMillis) This function is used to realize the palyer.

Method Detail

configure

public boolean **configure**(int timeOutMillis)

This function is used to configure the palyer.

Parameters:

int - Time to stay in that state

Returns:

boolean state of the player

---

close

public void **close**()

This function is used to close the palyer.

---

playToEndOfMedia

public boolean **playToEndOfMedia**(int timeOutMillis)

This function is used to detect end of media stream for the palyer.

Parameters:

int - Time to stay in that state

Returns:

boolean state of the stream

---

realize

public boolean **realize**(int timeOutMillis)

This function is used to realize the palyer.

Parameters:

int - Time to stay in that state

Returns:

boolean state of the player

---

prefetch

public boolean **prefetch**(int timeOutMillis)

This function is used for prefetching.

Parameters:

int - Time to stay in that state

Returns:

state of the palyer

---

controllerUpdate

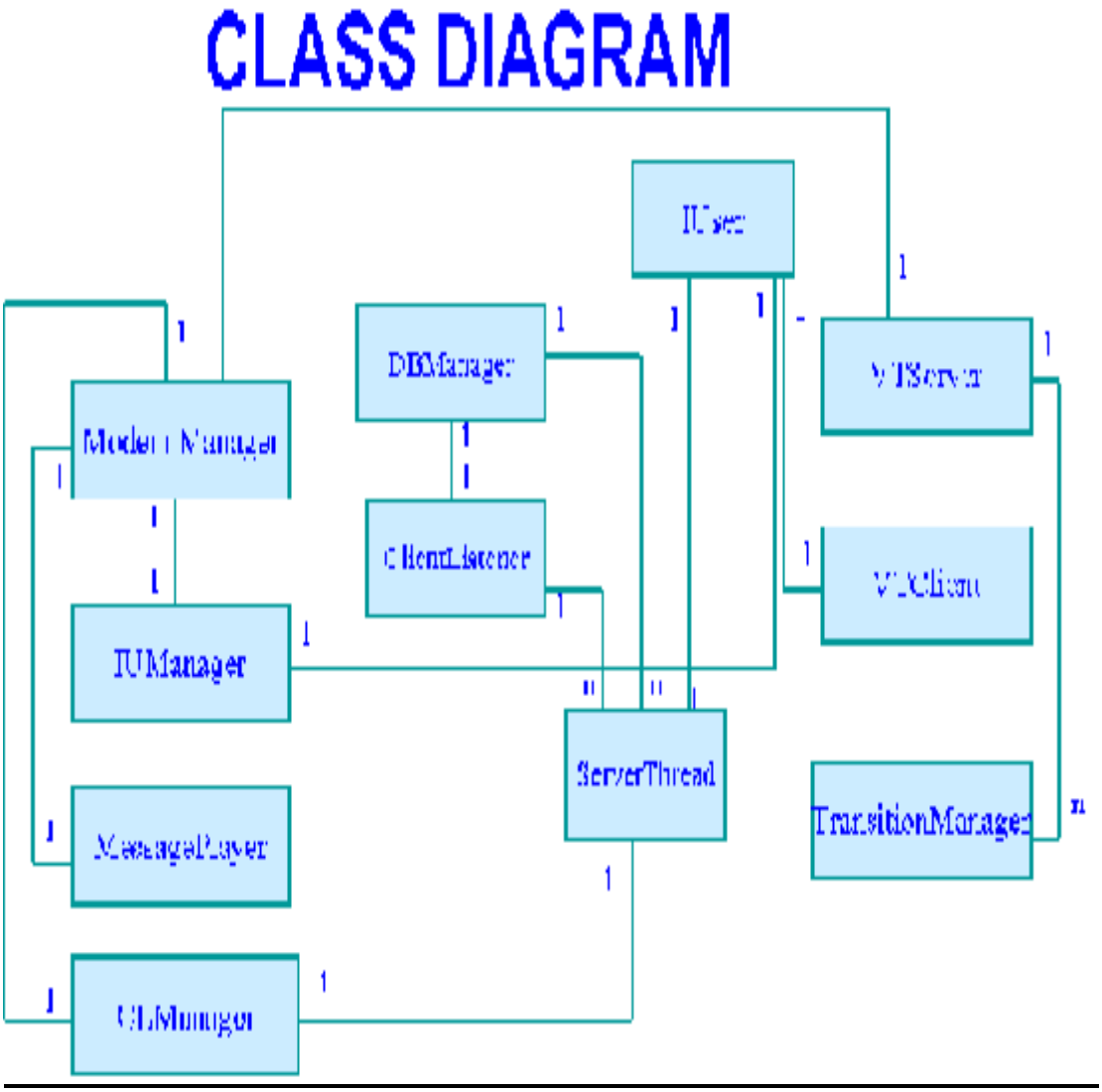
public void **controllerUpdate**(ControllerEvent ce)

This function is used for event handling.

Parameters:

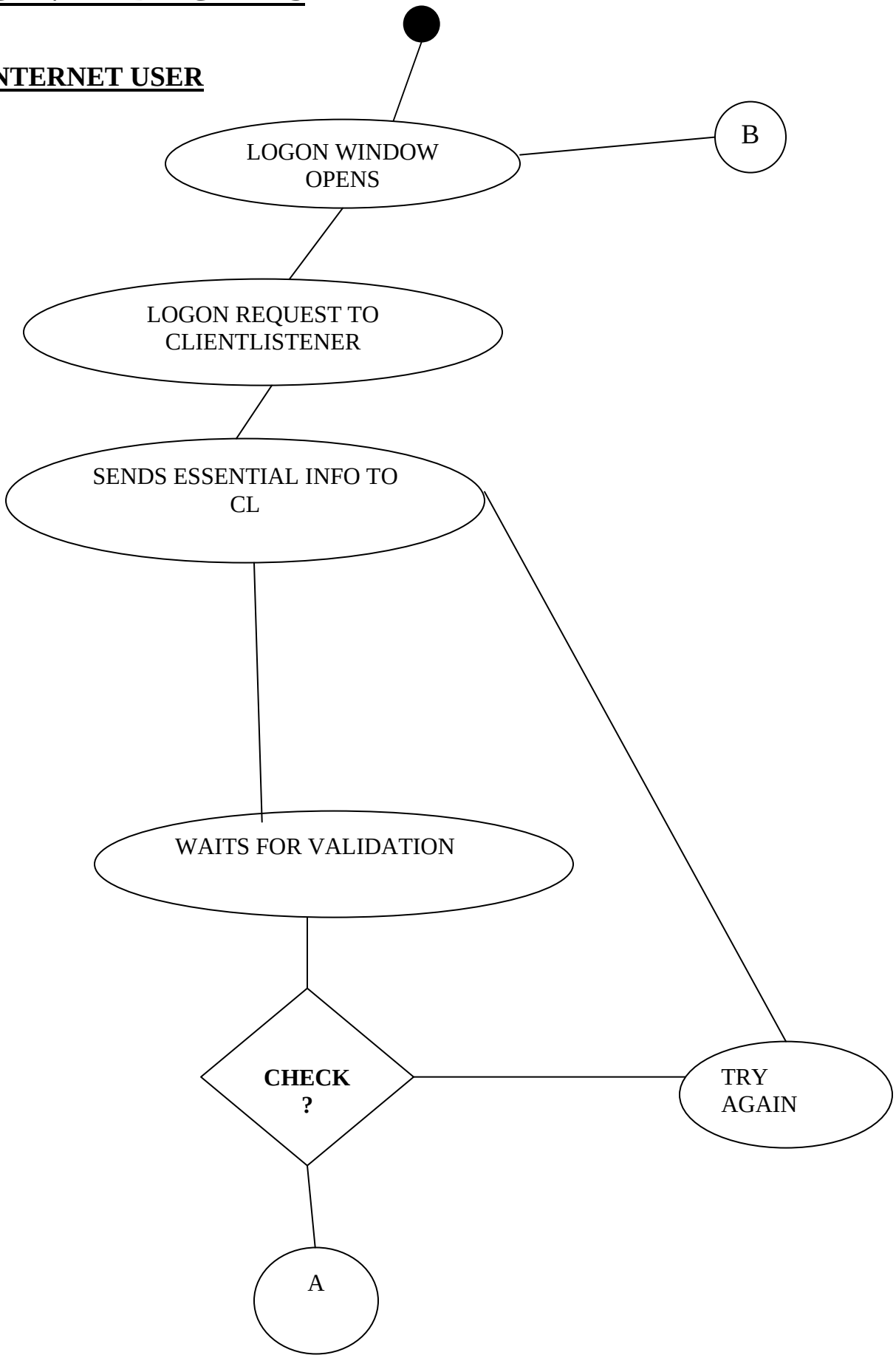
ce - event to update the controller

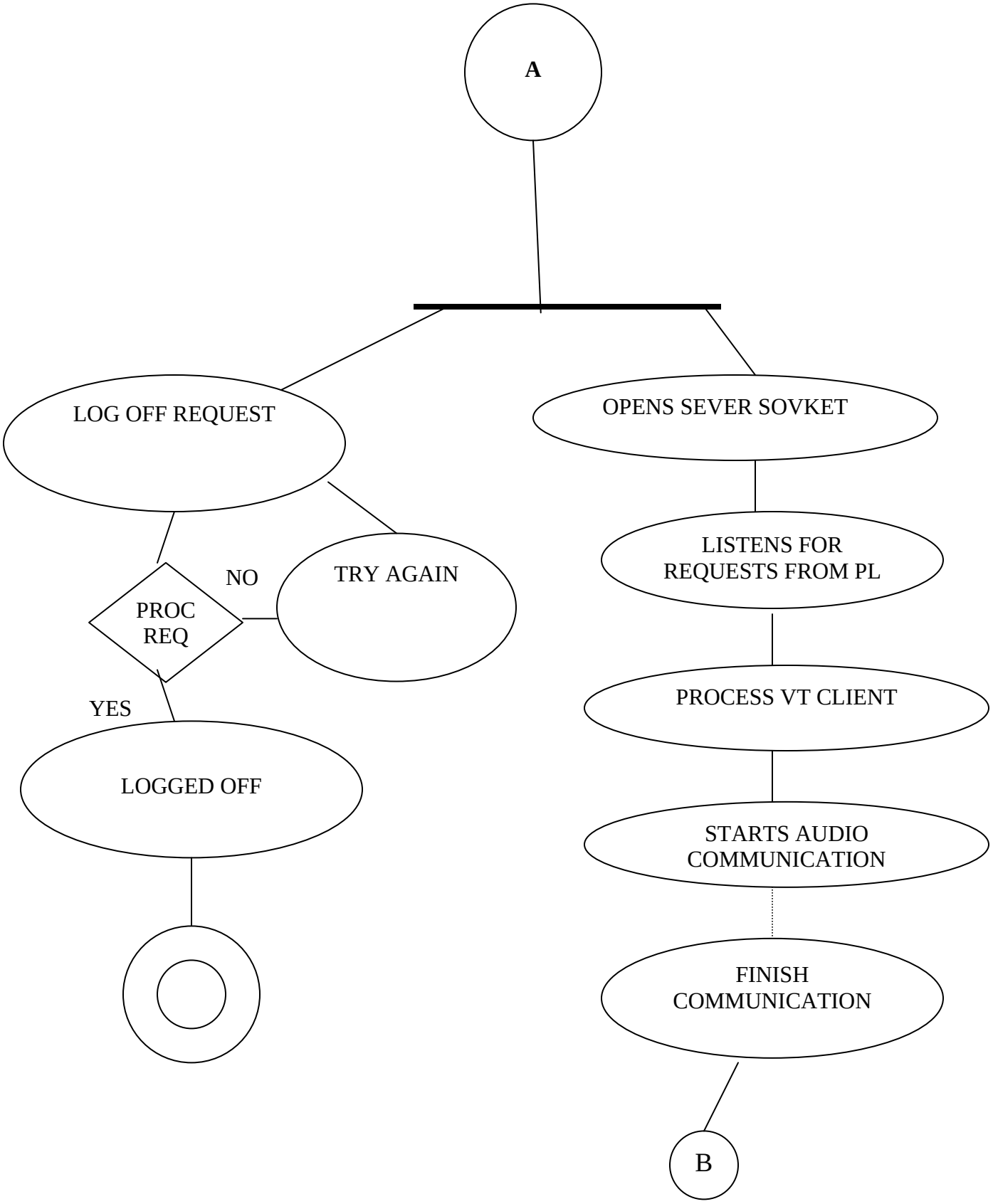
4.3 CLASS DIAGRAM



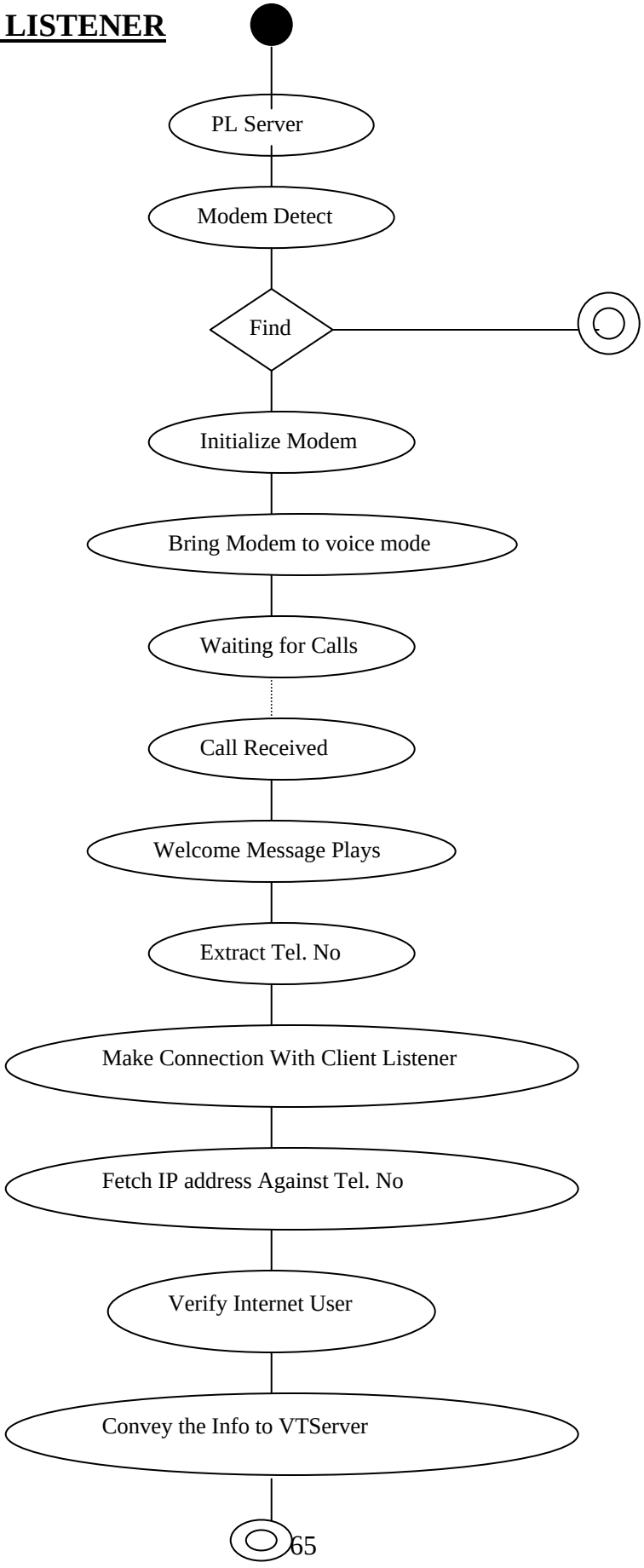
4.4 ACTIVITY DIAGRAMS

4.4.1 INTERNET USER

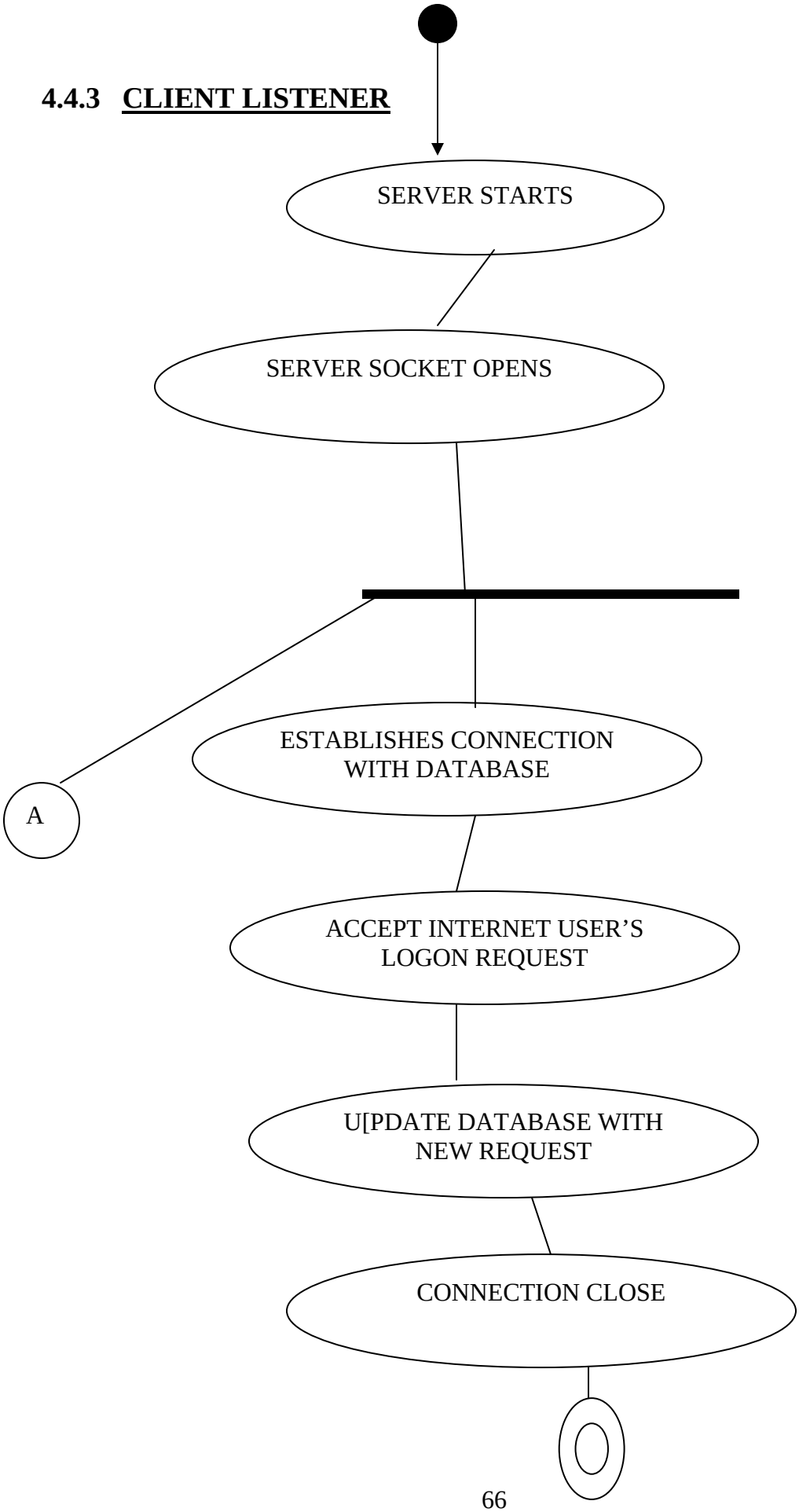


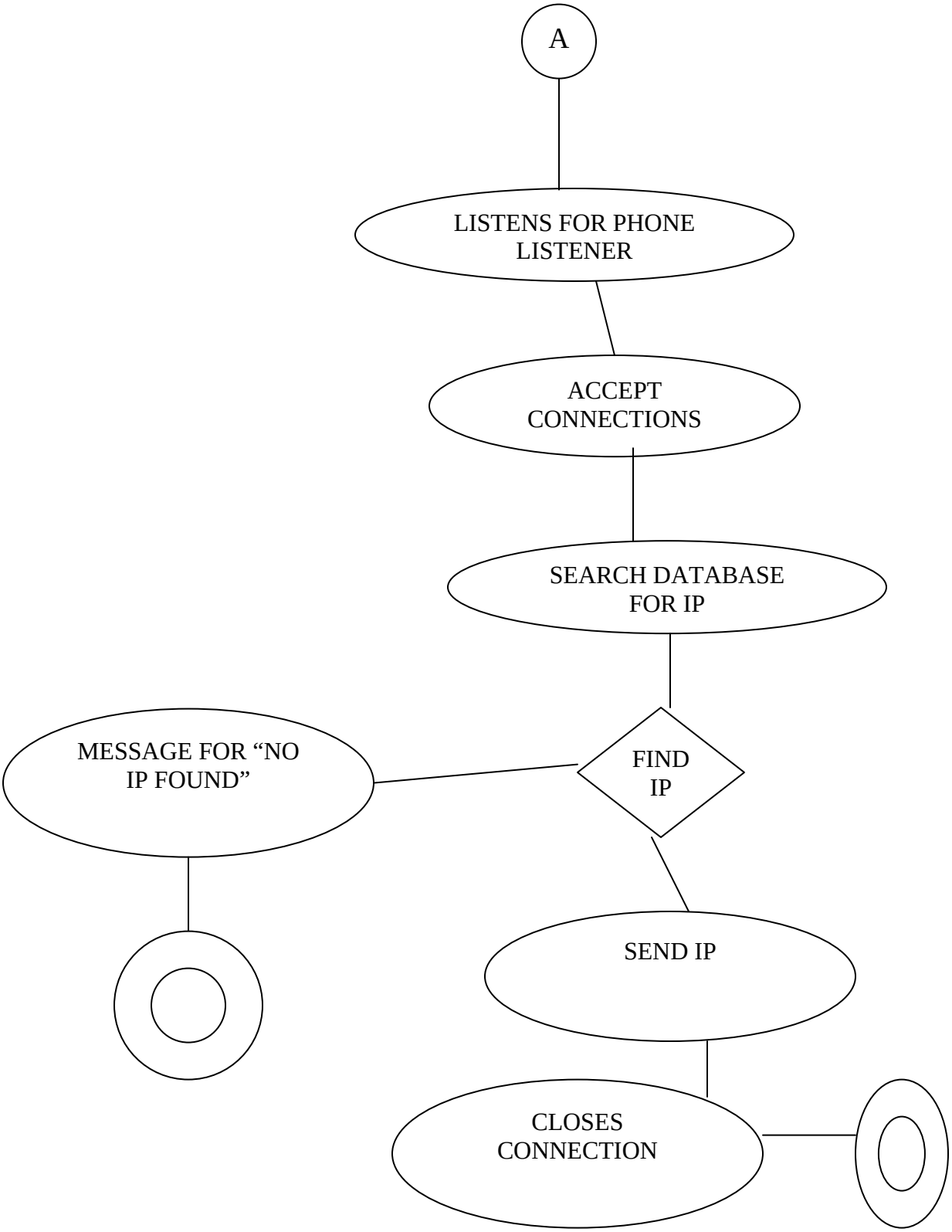

A
LOG OFF REQUEST
OPENS SEVER SOVKET
PROC REQ
NO
TRY AGAIN
YES
LOGGED OFF
LISTENS FOR REQUESTS FROM PL
PROCESS VT CLIENT
STARTS AUDIO COMMUNICATION
FINISH COMMUNICATION
B

4.4.2 PHONE LISTENER



4.4.3 CLIENT LISTENER




A
LISTENS FOR PHONE LISTENER
ACCEPT CONNECTIONS
SEARCH DATABASE FOR IP
FIND IP
MESSAGE FOR “NO IP FOUND”
SEND IP
CLOSES CONNECTION

5. OVERVIEW OF AT COMMAND

The modem may be configured in response to AT voice commands to provide enhanced Adaptive Differential Pulse Code Modulation (ADPCM) coding and decoding for the compression and decompression of digitized voice. ADPCM compression supports the efficient storage of voice messages, while optional coder silence deletion and decoder silence interpolation significantly increase compression rates. The ADPCM Voice Mode Supports three sub modes once a voice connection is established (see #CLS command): Online Voice Command Mode, Voice Receive Mode, and Voice Transmit Mode.

5.1 VOICE SUBMODES

5.1.1 ONLINE VOICE COMMAND MODE

Online Voice Command Mode is the default Voice sub mode entered when the #CLS=8 command is issued, and may also be entered from Voice Receive Mode or Voice Transmit Mode. Entry Into Online Voice Command Mode is indicated to the DTE via the VCON message, after which AT commands can be entered without aborting the telephone line connection.
If the modem is the answerer, it enters Online Voice Command Mode immediately after going off-hook, and can report instances of DTMF tones

and calling tones to the DTE. If the modem is the originator, it enters Online Voice Command Mode based on detection of the ring back cadence going away, upon expiration at the ring back never came timer, or upon detection of answer tone, and the modem can report DTMF tones, answer tones, busy tone, and dial tone to the DTE.

(Note that DTMF tone reporting is supported in this mode if DTMF reporting is enabled via the #VTD command.)

When this mode is entered as a result of going off-hook with the D or A command, VCON is always sent to the DTE, after which the modem accepts commands. If this mode is entered from Voice Transmit Mode, the DTE has issued the <DLE><ETX>, and the modem responds with VCON. If this mode is entered from the Voice Receive Mode because of a key abort, the modem issues the <DLE><ETX> followed by VCON.

If the #VLS command has switched in a handset or other device in place of the telephone line, Online Voice Command Mode is immediately entered, whereas if the telephone line is selected, a physical connection with another station must occur before entering this mode.

5.1.2 VOICE RECEIVE MODE

Voice Receive Mode is entered when the DTE issues the #VRX command because it wants to receive voice data. This typically occurs when either recording a greeting message, or when recording voice messages from a remote station.

In Voice Receive Mode, voice samples from the modem analog-to-digital converter (ADC) are sent to the ADPCM codec for compression, and can

then be read by the host. AT commands control the codec bits-per-sample rate and select (optional) silence deletion including adjustment at the silence detection period.

In this mode, the modem detects and reports DTMF, dial tone, busy tone cadence, and inactivity (periods of silence) as enabled by the #VTD and #VSS commands, respectively. The modem can exit the Voice Receive Mode only via a DTE Key Abort, or via Dead man Timer expiration (S30).

5.1.3 VOICE TRANSMIT MODE

Voice Transmit Mode is entered when the DTE issues the #VTX command because it wants to transmit voice data. In this mode, the modem continues to detect and report DTMF and calling tones if enabled by the #VTD command. This mode is typically used when playing back greeting messages or previously received/recorded messages. In this mode, voice decompression is provided by the codec, and decompressed data is reconstituted into analog voice by the DAC at the original voice compression quantization sample-per-bits rate. Optional silence interpolation is enabled if silence deletion was selected for voice compression.

5.2 VOICE CAPABILITIES

5.2.1 CALL ESTABLISHMENT - ANSWER

For most call originations, it is known ahead of time what type of call is being attempted, and it is acceptable to disconnect if the remote side of the

connection does not cooperate. In this case, the modem can be configured ahead of time with the existing +FCLASS (and +FAA) or the #CLS command to be a data, fax, or voice modem. For Data and Fax Modes, the modem subsequently either succeeds with the desired type of connection, or eventually hangs up. For the Voice Mode, the DTE has the option of hanging up if there are indications that the remote station has not answered in voice, thus implementing a directed originate for voice. The following are the three connection type choices:

5.2.1.2 VOICE

The modem dials and reports call progress to the DTE, which reduces to reporting NO DIALTONE, or BUSY. The modem allows the DTE to program a time period, which if elapsed after any ring back is detected, forces the modem to assume the remote has gone off-hook. A secondary time period (safety valve) can define a maximum elapsed time after dialing for receiving no ring back before the modem assumes that the remote has gone off-hook. This safety valve is devised in case the remote picks up the telephone before any ring back is generated, and no other tones are detected. In this mode, the modem is attempting to make a voice connection only and therefore, while waiting for ring back to disappear, it is also feasible to disconnect upon detection something which is definitely not Voice from the remote, such as any answer tone. The modem provides detection of "ring back" went away or never came.

5.2.1.3 FAX CAPABILITIES

The modem dials and reports call progress to the DTE as in all modes. A fax Class 1 or Class 2 handshake is pursued according to the current configuration.

5.2.1.4 DATA

The modem dials and reports call progress to the DTE as in all modes. A data handshake is pursued according to the current configuration. Adaptive Originate (Dial with Voice/Data/Fax Discrimination). The DTE may wish to originate a call, which adapts to the remote answerer. For instance, the user may wish to send a voice message if a human picks up the telephone, but a facsimile if a fax machine answers. The modem can facilitate this type of adaptive originate by extending what it does for the directed originate modes. Alter determining that the remote station has picked up the line, the modem goes back to Online Voice Command Mode, thus terminating the "connecting state." Once in this mode, the modem reports what it receives from the answerer via specific result codes to the DTE. The DTE can then have the option of pursuing a data, fax, or voice connection.

5.2.2 CALL ESTABLISHMENT - ANSWER

If the DTE wants to be only one kind of answerer (i.e., voice, fax, or data), it can configure the modem to answer exclusively in the chosen mode.

5.2.2.1 VOICE

The modem is configured to answer in Voice Mode only and assumes the caller will cooperate. After going off-hook, the voice VCON is issued, no answer tone is generated, and the modem is immediately placed In Online Voice Command Mode. The DTE typically responds by sending a greeting message of some type, and DTMF tone recognition/reporting can be enabled. Eventually, an Incoming voice message can be recorded by the host. (Unpredictable results occur if the caller is not prepared for a voice call.)

5.2.2.2 FAX CAPABILITIES

The modem is configured to answer in Class 1 or Class 2 Fax Mode only, and it assumes the caller is going to cooperate. This configuration has the effect of disabling Voice Mode, forcing +FCLASS to either 1 or 2, and forcing both +FAA and +FAE to 0.

5.2.2.3 DATA

The modem is configured to answer in Data Mode only and assumes the caller is going to cooperate. This configuration has the effect of disabling Voice Mode, forcing +FCLASS=0, and forcing both +FAA and +FAE to 0.

5.2.3 ADAPTIVE ANSWER (ANSWER WITH VOICE/DATA/FAX DISCRIMINATION)

In normal operation, it is desirable for a modem supporting fax and voice to provide the ability to discriminate between the two when answering unsolicited or unattended calls. (It is most often the case that a fax is received or a Voice message recorded when nobody is present.)

5.2.3.1 DATA/FAX DISCRIMINATION

If the DTE wishes to allow for a data or fax call, the +FCLASS and +FAA or +FAE commands can be configured for adaptive answer between data and Class 1 or Class 2 fax

5.2.3.2 VOICE/FAX DISCRIMINATION

This is the most important discrimination capability needed from the users standpoint. The modem must be configured for Voice (#CLS=8), causing the modem to enter Online Voice Command Mode immediately upon going off-hook. In Voice Mode, the DTE automatically receives indications of DTMF tones and Calling Tones. The DTE can now switch to Voice Transmit Mode in order to play a greeting message, perhaps one which instructs the caller how to enter specific DTMF sequences to switch modes. The DTE can then react to the response, or the lack thereof, to such a message. The modem supports switching to a Class 1 or Class 2 answer mode by virtue of the #CLS=1 or 2 commands, and if such a switch is made and fails, the modem reports the failure but does not hang up, allowing the DTE further experimentation time. If the user wishes to switch to Class 1 or 2, but also wants the DTE to indeed hang up the line if the fax fails, the

+FCLASS command should be used instead of the #CLS command. The only difference between these commands is that issuing +FCLASS cancels the modems memory of voice, where as #CLS causes the modem to remain off-hook, even if a fax or data handshake fails, until it receives an H command.

5.2.3.3VOICE/DATA/FAX DISCRIMINATION

The DTE can try data modem operation after an answer by changing the #CLS setting to 0. A data handshake attempt can be added based upon DTMF responses or lack thereof.

5.3 VOICE DATA TRANSFER

A significant area of concern when handling the transfer of voice data is the data transfer rate on the modem/DTE interface. Data transfer rates can be expressed as the number of interrupts which must be serviced per time period to keep up. This is a function of the sampling rate and compression method (if any) used by the modem, and the DTE interface speed required to handle the data flow on the telephone line side.

The modem can detect specific tones and other status information, and report these to the DTE while in any of the three voice sub modes. The modem simultaneously looks for 1300 and 1100 Hz calling tones when answering, and for CCITT and Bell answer tones when originating. The modem can also detect dial or busy tones in any of the three voice sub modes. All detected tones, as well as certain other statuses addressed such as silence and "teleset off-hook" (i.e., handset off-hook) are reported as

shielded codes. When in Online Voice Command Mode or Voice Transmit Mode, the codes are sent to the DTE immediately upon verification by the modem of the associated tone, status, or cadence. In this mode, the 2-character code is not buffered, nor does the DTE have the ability to stop the code with flow control. If the DTE has started (but not completed) sending any AT command, the Tone Monitoring function is disabled until the command has been received and processed. The modem can discriminate between single and multiple DTMF tones received. If calling tone, dial tone, busy tone, or answer tone is detected, this detection is reported repeatedly (at reasonable intervals) if the DTE takes no action, and the tone continues to be detected.

5.4 TABLE SHIELDED CODES SENT TO THE DTE

CODE SENT TO DTE	MEANING
<DLE>0-<DLE>9	DTMF. Digits 0 through 9, *, #, or A through D detected
<DLE>*,<DLE>#	by the modem, i.e., user has pressed a key on a local or <DLE>A-<DLE>D remote telephone. The modem sends only one <DLE> code per DTMF button pushed.
<DLE>a	Answer Tone (CCITT). Send to the DTE when the V.25/T.30 2100 Hz Answer Tone (Data or Fax) is detected. If the DTE fails to react to the code, and the modem continues

	to detect Answer tone, the code is repeated as often as once every half second.
<DLE>b	Busy. Sent in Voice Receive Mode when the busy cadence s detected, after any remaining data in the voice in receive buffer. The modem sends the busy <DLE>b code every 4 seconds if busy continues to be detected and the DTE does not react. This allows the DTE the flexibility of ignoring what could be a false busy detection.

5.5 VOICE PLAYBACK

To effect playback of a message recorded via a handset or microphone, or of a message recorded during a voice call, the DTE must configure the modem for Voice Mode (#CLS=8) and select the proper relay setup (#VLS) to instruct the modem whether to use the handset or speaker. The modem responds to the #VLS command by issuing a relay activate command to select the input device. The hardware must provide a means of selecting a handset and/or microphone instead of the telephone line, as this input device. When a device other that the telephone line is selected, the modem immediately enters Online Voice Command Mode (indicated by VCON). DTMF detection is thus enabled as soon as the DTE selects the device, such as a handset, although the user still needs to physically pick up the telephone before he can issue DTMF tones. Once selected, however, the

user can indeed pick up the telephone and "press buttons." Even if the DTE has not entered Voice Receive or Transmit Modes (#VTX or #VRX), these DTMF tones are delivered via shielded codes, identically to when a physical telephone connection exists but the DTE has not yet commanded receive nor transmit.

When the DTE decides to play the message, it issues the #VTX command, and the modem immediately switches to Voice Transmit Mode. Since the speaker or handset is already switched in, the modem immediately issues the CONNECT message indicating that the modem is in Voice Transmit Mode and is expecting Voice data from the DTE. A subsequent <DLE><ETX> has to be issued to switch back to Online Voice Command Mode.

5.6 VOICE CALL TERMINATION

5.6.1 LOCAL DISCONNECT

The DTE can disconnect from a telephone call by commanding a mode change to Online Voice Command Mode (if not already in it), and by issuing the H command.

5.6.2 REMOTE DISCONNECT DETECTION

When In Voice Receive Mode, the modem sends the proper shielded <DLE> code when loop break, dial tone, or busy tone is detected. The modem stays in Voice Receive Mode, however, until the DTE issues a key

abort to force Online Voice Command Mode. The DTE must issue the H command if it wishes to hang up.

5.7 MODE SWITCHING

5.7.1 VOICE TO FAX

If the modem is in Online Voice Command Mode (i.e. it has gone off-hook with #CLS=8 in effect), the DTE can attempt a fax handshake by setting #CLS=1 or #CLS=2 followed by the A or D command corresponding to fax receive or send. This has the effect of beginning a fax Class 1 or Class 2 handshake (see #CLS command).

5.7.1.1 UNSUCCESSFUL FAX CONNECTION ATTEMPT TO VOICE

A fax handshake which does not succeed, attempted as the result of the DTE modifying the #CLS setting from voice (8) to fax (1 or 2) does not result in the modem hanging up, allowing the DTE the flexibility of commanding a switch back to Voice Mode with #CLS=8.

5.7.2 VOICE TO DATA

If the modem is in the Online Voice Command Mode the DTE can attempt a data

handshake by setting #CLS=0 followed by the A or D command. This has the effect of beginning a Data Mode handshake according to the current Data Mode S-register and command settings.

5.7.2.1 UNSUCCESSFUL DATA CONNECTION ATTEMPT TO VOICE

A data handshake which does not succeed attempted as the result of the DTE modifying the #CLS setting from voice (8) to data (0), does not result in the modem hanging up, allowing the DTE the flexibility of commanding a switch back to Voice Mode with #CLS=8.

5.8 CALLER ID

The modem supports Caller ID by passing the information received in Bell 202 FSK format to the DTE after the first RING detect. The modem supports both formatted and unformatted reporting of Caller ID information received in ICLID (Incoming Call Line ID) format as supported in certain areas of the U.S. and Canada. The DTE enables this feature via the #CID command.

5.9 AT VOICE COMMAND SUMMARY

Table provides a complete summary of the AT voice commands described in detail in following sections

5.9.1 GLOBAL AT COMMAND SET EXTENSIONS

The AT commands in the following section are global meaning that they can be issued in any appropriate mode (i.e., any #CLS setting).For consistency, the command set is divided into action commands and

parameters (non-action commands). Those commands which are action commandsi.e., those which cause some change in the current operating behavior of the modem) are identified as such, and the remaining commands are parameters.

5.9.2 ATA - ANSWERING IN VOICE

The answer action command works analogously to the way it works in Data and Fax Modes except for the following:

1. When configured for Voice Mode (#CLS=8), the modem enters Online Voice Command Mode immediately after going off-hook. When the #CLS=8 command is issued, the modem can be programmed to look for 1100 and 1300 Hz calling tones (see #VTD), thus eliminating the need to do so as part of A command processing. After the VCON message is issued the modem re-enters Online Voice Command Mode while sending any incoming DTMF or calling Tone indications to the DTE
2. After answering in Voice Mode (#CLS=8) the DTE, as part of its call discrimination processing can decide to change the #CLS setting to attempt receiving a fax in Class 1 or to make a data connection. In such a case the DTE commands the modem to proceed with the data or fax handshake via the A command even though the modem is already off-hook.

5.9.3 VCON

Issued in Voice Mode (#CLS=8) immediately after going off-hook

Command	Function
A	Answering in Voice Mode.
D	Dial command in Voice Mode.
H	Hang up in Voice Mode.
Z	Reset from Voice Mode.
#BDR	Select baud rate (turn off auto baud).
#CID	Enable Caller ID detection and select reporting format.
#CLS	Select data fax or voice
#MDL?	Identify model.
#MFR?	Identify manufacturer.
#REV?	Identify revision level.
#VBQ?	Query buffer size.
#VBS	Bits per sample (ADPCM).
#VBT	Beep tone timer.
#VCI?	Identify compression method (ADPCM).
#VLS	Voice line select (ADPCM).
#VRA	Ringback goes away timer (originate).
#VRN	Ringback never came timer (originate).
#VRX	Voice Receive Mode (ADPCM).
#VSD	Silence deletion tuner (voice receive ADPCM).
#VSK	Buffer skid setting.
#VSP	Silence detection period (voice receive ADPCM)
#VSR	Sampling rate selection (ADPCM).
#VSS	Silence deletion tuner (voice receive)
#VTD	DTMF/tone reporting capability.
#VTX	Voice Transmit Mode (ADPCM).

5.9.4 ATD

Dial Command In Voice

The dial action command works analogously to the way it works in Data or Fax modes. When In Voice Mode (#CLS=8):

1. The modem attempts to determine when the remote has picked up the telephone line and once this determination has been made, the VCON message is sent to the DTE. This determination is initially made based upon ringback detection and disappearance. (See #VRA and #VRN commands.)

2. Once connected in Voice Mode the modem immediately enters the command state and switches to Online Voice Command Mode which enables unsolicited reporting of DTMF and answer tones to the DTE.

Parameters: Same as Data and Fax modes.

5.9.5 VCON

Issued in Voice Mode (#CLS=8) when the modem determines that the remote modem or handset has gone off-hook, or when returning to the Online Voice Command Mode. (See #VRA and #VRN.)

5.9.6 NO ANSWER

Issued in Voice Mode (#CLS=8) when the modem determines that the remote has not picked up the line before the S7 timer expires.

5.9.7 ATH

Hang Up In Voice

This command works the same as in Data and Fax modes by hanging up (disconnecting) the telephone line. There are, however, some specific considerations when in Voice Mode:

1. The H command forces #CLS=0 but does not destroy any of the voice parameter settings such as #VBS, #VSP, etc. Therefore if the DTE wishes to issue an H command and then pursue another voice call it must issue a subsequent #CLS=8 command, but it needn't reestablish the voice parameter settings again unless a change in the settings is desired.

2. The #BDR setting is forced back to 0, re-enabling auto baud.

3. If the #VLS setting is set to select a device which is not, or does not include the telephone line (such as a local handset or microphone), the H command deselects this device and reselects the normal default setting (#VLS=0). Normally, the DTE should not issue the H command while connected to a local device each as a handset, because merely selecting this device results in VCON. The normal sequence of terminating a session with

such a device is to use the #VLS command to select the telephone line, which by definition makes sure it is on-hook.

5.9.8 ATZ

Reset from Voice Mode

This command works the same as in Data and Fax modes. In addition, the Z command resets all voice related parameters to default states, forces the #BDR=0 condition (autobaud enabled), and forces the telephone line to be selected with the handset on-hook. No voice parameters are stored in NVRAM so the profile loaded does not affect the voice aspects of this command.

5.9.9 #BDR

Select Baud Rate (Turn off Autobaud)

This command forces the modem to select a specific DTE/modem baud rate without further speed sensing on the interface. When a valid #BDR=n command is entered, the OK result code is sent at the current assumed speed. After the OK has been sent, the modem switches to the speed indicated by the #BDR=n command it has just received.

When In Online Voice Command Mode and the #BDR setting is nonzero (no autobaud selected), the modem supports a full duplex DTE interface. This means that the DTE can enter commands at any time, even if the modem is in the process of sending a shielded code indicating DTMF

detection to the DTE. When in Online Voice Command Mode and the #BDR setting is zero (autobaud selected), shielded code reporting to the DTE is disabled. [Note that when #BDR has been set nonzero, the modem employs the S30 Deadman Timer, and this timer starts at the point where #BDR is set nonzero. If this period expires (nominally 60 seconds) with no activity on the DTE interface, the modem reverts to #BDR=0 and #CLS=0.

5.9.10 #BDR?

Returns the current setting of the #BDR command as an ASCII decimal value in result code format.

5.10 AT#V COMMANDS ENABLED ONLY IN VOICE MODE (#CLS=8)

The commands described in the following subsection are extensions to the command set which the modem recognizes only when configured for Voice Mode with the #CLS=8 command.

#VBQ? Query Buffer Size

#VBQ? Returns the size of the modem voice transmit and Voice receive buffers.

#VBS Bite Per Sample (Compression Factor)

#VBS? Returns the current setting of the #VBS command as an ASCII decimal value in result code format.

#VBS=? Returns "2,3,4", which are the ADPCM compression bits/sample rates available. These bits/sample rates are correlated with the #VCI?

query command response which provides the single compression method available.

#VBS=2 Selects 2 bits per sample.

#VBS=3 Selects 3 bits per sample.

#VBS=4 Selects 4 bits per sample.

#VBT Beep Tone Timer

5.11 DEVICE TYPES SUPPORTED BY #VLS

5.11.1 ASCII DIGIT DEVICE TYPE AND CONSIDERATIONS

0 Telephone Line with Telephone handset. This is the default device selected. In this configuration, the user can pick up a handset which is connected to the same telephone line as the modem, and * record both sides of a conversation with a remote station. The modem currently supports one telephone line/handset, which is in the first position of the #VLS=? response. (Note that the modem can interface to multiple telephone lines by having "0"'s in multiple positions in the #VLS? response.) I telephone line is selected, the modem must be on-hook or it hangs up. The OK message is generated.

1 Transmit/Receive Device (other than telephone line). This is a handset, headset, or speaker-phone powered directly by the modem. When such a device is selected, the modem immediately enters Online Voice Command Mode, DTMF monitoring is

enable if applicable, and the VCON response is sent. The modem supports one such device as the second device listed in the #VLS=? response.

2 Transmit Only Device. Normally, this is the onboard speaker. When this device is selected, the modem immediately enters Online Voice Command Mode, and the VCON response is sent. The modem supports selection of the internal speaker as the third device listed in #VLS=? response.

3 Receive Only Device. Normally, this is a microphone. When such a device is selected, the modem immediately enters Online Voice Command Mode, DTMF monitoring is enabled if applicable, and the VCON response is sent. The modem supports one microphone as the fourth element returned in the #VLS=? response.

4 Telephone line with Speaker ON and handset. This device type can be used to allow the DTE to select the telephone line/handset (if picked up) with the modem speaker also turned ON. This can be used by the DTE to allow the user to monitor an incoming message as it is recorded.

5.12 S-REGISTERS

The following S-register is global, meaning that it can be set in any appropriate mode (i.e., any #CLS setting).

S30 - Deadman (Inactivity) Timer

Range: n = 0 - 255

Default: 0 (OFF, which means DTE should usually set it to some value for Voice)

Command options:

S30=0 Dead man timer off. No matter how long it might continue, the modem never spontaneously hangs up the telephone line or switches to auto baud mode as a result of inactivity.

S30=1 to 255 This is the period of time (in seconds), which if expired causes the modem to hang up the telephone line if it is off-hook and no data has passed during the period. The timer is also active whenever the #BDR setting is non-zero. In order to avoid a state where speed sense is disabled (even though the PC can crash, come back up, and try to issue commands at what should be a supported speed), the inactivity time-out occurs if there is no data passed on the DTE interface within the S30 period, even if the modem is on-hook. DTE software must not select a nonzero setting for #BDR until it is ready to establish a telephone call or virtual connection to a speaker or microphone. When there is an inactivity time out with #CLS=8, the modem always forces #CLS=0 and #BDR=0.

5.13 RESULT CODES FOR VOICE OPERATION

VCON is sent when the modem is configured for Voice (#CLS=8), or when after answering or originating a call, the modem enters the Online Voice Command Mode for the first time. Typically, this is immediately

after an off-hook in answer mode, and after ringback ceases in originate mode. VCON is also sent when the DTE requests a switch from Voice Transmit Mode to Online Voice Command Mode by issuinq a <DLE><ETX> to the modem, or when the DTE requests a switch from Voice Receive Mode to Online Voice Command Mode via the key abort.
CONNECT CONNECT is sent when switching from the Online Voice Command Mode to either Voice Receive Mode via the #VRX command, or to Voice Transmit Mode via the #VTX command. This message is sent to the DTE to inform it that it may begin receiving or sending ADPCM data.

6 OVERVIEW OF JCOMM

6.1 JAVAX.COMM EXTENSION PACKAGE

There are three levels of classes in the Java communications API:

- High-level classes like CommPortIdentifier and CommPort manage access and ownership of communication ports.
- Low-level classes like SerialPort and ParallelPort provide an interface to physical communications ports. The current release of the Java communications API enables access to serial (RS-232) and parallel (IEEE 1284) ports.
- Driver-level classes provide an interface between the low-level classes and

the underlying operating system. Driver-level classes are part of the implementation but not the Java communications API. They should not be used by application programmers.

The javax.comm package provides the following basic services:

- Enumerate the available ports on the system. The static method CommPortIdentifier.getPortIdentifiers returns an enumeration object that contains a CommPortIdentifier object for each available port. This CommPortIdentifier object is the central mechanism for controlling access to a communications port.
- Open and claim ownership of communications ports by using the high level methods in their CommPortIdentifier objects.
- Resolve port ownership contention between multiple Java applications. Events are propagated to notify interested applications of ownership contention and allow the port's owner to relinquish ownership. PortInUseException is thrown when an application fails to open the port.
- Perform asynchronous and synchronous I/O on communications ports. Low-level classes like SerialPort and ParallelPort have methods for managing I/O on communications ports.
- Receive events describing communication port state changes. For example, when a serial port has a state change for Carrier Detect, Ring Indicator, DTR, etc. the SerialPort object propagates a SerialPortEvent that describes the state change.

A Simple Reading Example

- SimpleRead.java opens a serial port and creates a thread for asynchronously reading data through an event callback technique.

A Simple Writing Example

- SimpleWrite.java opens a serial port for writing data.

6.2 SERIAL SUPPORT WITH JAVAX.COMM PACKAGE

Sun's JavaSoft division provides support for RS-232 and parallel devices with standard extensions.

SUMMARY

One of the most popular interfaces on a PC is the serial port. This interface allows computers to perform input and output with peripheral devices. Serial interfaces exist for devices such as modems, printers,bar code scanners, smart card readers, PDA interfaces, and so on. Sun's JavaSoft division recently has made available the javax.comm package to add serial support to Java. This package provides support for serial and parallel devices using traditional Java semantics such as streams and events. In order to communicate with a serial device using a serial port on a host computer from a Java application or applet, an devices connected to your serial port. In addition, the API provides a complete set of options for setting all of the parameters associated with serial and parallel devices.This article focuses on

how to use javax.comm to communicate with a serial device based on RS-232; discusses what the javax.comm API does and does not provide; and offers a small example program that shows you how to communicate to the serial port using this API. We will end with a brief discussion of how this API will work with other device drivers, and also go over the requirements for performing a native port of this API to a specific OS. (2,700 words)

The Java Communications (a.k.a. javax.comm) API is a proposed standard extension that enables authors of communications applications to write Java software that accesses communications ports in a platform-independent way. This API may be used to write terminal emulation software, fax software, smart-card reader software, and so on. Developing good software usually means having some clearly defined interfaces. The high-level diagram of the API interface layers are shown in this figure.

In this article we will show you how to use javax.comm to communicate with a serial device based on RS-232. We'll also discuss what the javax.comm. API provides and what it doesn't provide. We'll present a small example program that shows you how to communicate to the serial port using this API. At the end of the article we'll briefly detail how this javax.comm. API will work with other device drivers, and we'll go over the requirements for performing a native port of this API to a specific OS.

Unlike classical drivers, which come with their own models of communication of asynchronous events, the javax.comm API provides an event-style interface based on the Java event model (java.awt.event package). Let's say we want to know if there is any new data sitting on the input buffer. We can find that out in two ways -- by polling or listening. With polling, the processor checks the buffer periodically to see if there is any new data in the buffer. With listening, the processor waits for an event

to occur in the form of new data in the input buffer. As soon as new data arrives in the buffer, it sends a notification or event to the processor.

Dialer management and modem management are additional applications that can be written using the javax.comm API. Dialer management typically provides an interface to the modem management's AT command interface. Almost all modems have an AT command interface. This interface is documented in modem manuals.Perhaps a little example will make this concept clear. Suppose we have a modem on COM1 and we want to dial a phone number. A Java dialer anagement application will query for the phone number and interrogate the modem. These commands are carried by javax.comm, which does no interpretation. To dial the number 918003210288, for example, the dialer management probably sends an "AT," hoping to get back an "OK," followed by ATDT918003210288. One of the most important tasks of dialer management and modem management is to deal with errors and timeouts.

GUI for serial port managemen Normally, serial ports have a dialog box that configures the serial ports, allowing users to set parameters suchas baud rate, parity, and so on. The following diagram depicts the objects involved in reading and/or writing data to a serial port from Java. Support for X, Y, and Z modem protocols. These protocols provide support error detection and correction.

The programming basics

Too often, programmers dive right into a project and code interactively with an API on the screen without giving any thought to the problem they are trying to solve. To avoid confusion and potential problems, gather the following information before you start a project. Remember,

programming devices usually requires that you consult a manual. Get the manual for the device and read the section on the RS-232 interface and RS-232 protocol.

Most devices have a protocol that must be followed. This protocol will be carried by the javax.comm API and delivered to the device. The device will decode the protocol, and you will have to pay close attention to sending data back and forth. Not getting the initial set-up correct can mean your application won't start, so take the time to test things out with a simple application. In other words, create an application that can simply write data onto the serial port and then read data from the serial port using the javax.comm. API. Try to get some code samples from the manufacturer. Even if they are in another language, these examples can be quite useful.Find and code the smallest example you can to verify that you can communicate with the device. In the case of serial devices, this can be very painful -- you send data to a device connected to the serial port and nothing happens. This is often the result of incorrect conditioning of the line. The number one rule of device programming (unless you are writing a device driver) is to make sure you can communicate with the device. Do this by finding the simplest thing you can do with your device and getting that to work. If the protocol is very complicated, consider getting some RS-232 line analyzer software.

This software allows you to look at the data moving between the two devices on the RS-232 connection without interfering with the transmission. Using the javax.comm API successfully in an application requires you toprovide some type of interface to the device protocol using the serial APIas the transport mechanism. In other words, with the exception of the simplest devices, there is usually another layer required to format thedata for

the device. Of course the simplest protocol is "vanilla" --meaning there is no protocol. You send and receive data with no interpretation.

6.3 OVERVIEW OF SUGGESTED STEPS FOR USING JAVAX.COMM

In addition to providing a protocol, the ISO layering model used for TCP/IP also applies here in that we have an electrical layer, followed by a very simple byte transport layer. On top of this byte transport layer you could put your transport layer. For example, your PPP stack could use the javax.comm API to transfer bytes back and forth to the modem. The role of the javax.comm layer is quite small when looked at in this context: Give the javax.comm API control of some of the devices. Before you use a device, the javax.comm API has to know about it. Open the device and condition the line. You may have a device that requires a baud rate of 115 kilobits with no parity. Write some data and/or read data following whatever protocol the device you are communicating with requires. For example, if you connect to a printer, you may have to send a special code to start the printer and/or end the job. Some PostScript printers require you to end the job by sending CTRL-D 0x03. Close the port.Initializing the javax.comm API registry with serial interface ports

The javax.comm API can only manage ports that it is aware of. The latest version of the API does not require any ports to be initialized. On start-up, the javax.comm API scans for ports on the particular host and adds them automatically. You can initialize the serial ports your javax.comm API can use. For devices that do not follow the standard naming convention Writing

and reading data For javax.comm, this is no different than any other read and write method call to the derived output stream.

For write:

```
try {
output.write ( outputArray, 0 , length );
```

For read:

```
try {
int b = input.read()
```

Closing the port:

Closing the port with javax.comm is no different than with other requests to close a device. This step is very important to javax.comm because it attempts to provide exclusive access. Multiplexing multiple users on a serial line requires a Multiplexor protocol.

```
try {
inout.close();
output.close();
} ...
```

6.4 CONCLUSION

The javax.comm API provides a modern disciplined approach to serial communications and will move Java into new application spaces, allowing devices like bar code scanners, printers, smart card readers, and hundreds of other serial devices to be connected with ease. The API is easy to use, as demonstrated by the example. It is also easy to port to new hardware platforms. The API has not been tested for high data rate and real time applications; therefore, developers looking to use the API in those types of environments should perform careful instrumentation with subsequent analysis of the code. In determining whether the API is suitable for high data rate or mission sensitive applications, look for the following:

- Characters lost on input
- Characters lost in output
- Frequency of flow control
- Time it takes to deliver an event
- Character processing times
- Block processing times

When we first started our series on smart cards, we were lucky if we understood a few native method calls to send bytes to serial devices. We end our smart card series with this article. The software APIs we have been discussing in this series come together from a device point of view. For example, a user developing an application for smart cards can write to some well-defined APIs such as OpenCard Framework or communicate directly using javax.comm -- or alternatively use

javax.smartcard, which in turn uses javax.comm. The javax.comm API facilitates the interfacing of serial and parallel devices to Java.

7. OVERVIEW OF VOICE TRAMSMISSION OVER INTERNET

7.1 UNDERSTANDING JMF

JavaTM Media Framework (JMF) provides a unified architecture and messaging protocol for managing the acquisition, processing, and delivery of time-based media data. JMF is designed to support most standard media content types, such as AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, and WAV. By exploiting the advantages of the Java platform, JMF delivers the promise of "Write Once, Run AnywhereTM" to developers who want to use media such as audio and video in their Java programs. JMF provides a common cross-platform Java API for accessing underlying media frameworks. JMF implementations can leverage the capabilities of the underlying operating system, while developers can easily create portable Java programs that feature time-based media by writing to the JMF API.

With JMF, you can easily create applets and applications that present, capture, manipulate, and store time-based media. The framework enables advanced developers and technology providers to perform custom processing of raw media data and seamlessly extend JMF to support additional content types and formats, optimize handling of supported formats, and create new presentation mechanisms.

High-Level Architecture Devices such as tape decks and VCRs provide a familiar model for recording, processing, and presenting time-based media. When you play a movie using a VCR, you provide the media stream to the VCR by inserting videotape. The VCR reads and interprets the data on the tape and sends appropriate signals to your television and speakers.

JMF uses this same basic model. A data source encapsulates the media stream much like videotape and a player provides processing and control mechanisms similar to a VCR. Playing and capturing audio and video with JMF requires the appropriate input and output devices such as microphones, cameras, speakers, and monitors. Data sources and players are integral parts of JMF's high-level API for managing the capture, presentation, and processing of time-based media. JMF also provides a lower-level API that supports the seamless integration of custom processing components and extensions. This layering provides Java developers with an easy-to-use API for incorporating time-based media into Java programs while maintaining the flexibility and extensibility required to support advanced media applications and future media technologies.

7.1.1 TIME MODEL

JMF keeps time to nanosecond precision. A particular point in time is typically represented by a Time object, though some classes also support the specification of time in nanoseconds. Classes that support the JMF time model implement Clock to keep track of time for a particular media stream. The Clock interface defines the basic timing and synchronization operations that are needed to control the presentation of media data.

A Clock uses a TimeBase to keep track of the passage of time while a media stream is being presented. A TimeBase provides a constantly ticking time source, much like a crystal oscillator in a watch. The only information that a TimeBase provides is its current time, which is referred to as the time-base time. The time-base time cannot be stopped or reset. Time-base time is often based on the system clock. A Clock object's media time represents the current position within a media stream--the beginning of the stream is media time zero, the end of the stream is the maximum media time for the stream. The duration of the media stream is the elapsed time from start to finish--the length of time that it takes to present the media stream. (Media objects implement the Duration interface if they can report a media stream's duration.) To keep track of the current media time, a Clock uses: The time-base start-time--the time that its Time Base reports when the presentation begins. The media start-time--the position in the media stream where presentation begins. The playback rate--how fast the Clock is running in relation to its Time Base. The rate is a scale factor that is applied to the TimeBase. For example, a rate of 1.0 represents the normal playback rate for the media stream, while a rate of 2.0 indicates that the presentation will run at twice the normal rate. A negative rate indicates that the Clock is running in the opposite direction from its TimeBase--for example, a negative rate might be used to play a media stream backward. When presentation begins, the media time is mapped to the time-base time and the advancement of the time-base time is used to measure the passage of time. During presentation, the current media time is calculated using the following formula:

MediaTime = MediaStartTime + Rate(TimeBaseTime - TimeBaseStartTime)

When the presentation stops, the media time stops, but the time-base time continues to advance. If the presentation is restarted, the media time is remapped to the current time-base time. Managers The JMF API consists mainly of interfaces that define the behavior and interaction of objects used to capture, process, and present time-based media. Implementations of these interfaces operate within the structure of the framework. By using intermediary objects called managers, JMF makes it easy to integrate new implementations of key interfaces that can be used seamlessly with existing classes. JMF uses four managers: Manager--handles the construction of Players, Processors, DataSources, and DataSinks. This level of indirection allows new implementations to be integrated seamlessly with JMF. From the client perspective, these objects are always created the same way whether the requested object is constructed from a default implementation or a custom one. PackageManager--maintains a registry of packages that contain JMF classes, such as custom Players, Processors, DataSources, and DataSinks. CaptureDeviceManager--maintains a registry of available capture devices. PlugInManager--maintains a registry of available JMF plug-in processing components, such as Multiplexers, Demultiplexers, Codecs, Effects, and Reindeers.

To write programs based on JMF, you'll need to use the Manager create methods to construct the Players, Processors, DataSources, and DataSinks for your application. If you're capturing media data from an input device, you'll use the CaptureDeviceManager to find out what devices are available and access information about them. If you're interested in controlling what processing is performed on the data, you might also query the PlugInManager to determine what plug-ins have been registered. If you extend JMF functionality by implementing a new plug-in, you can register it

with the PlugInManager to make it available to Processors that support the plug-in API. To use a custom Player, Processor, DataSource, or DataSink with JMF, you register your unique package prefix with the PackageManager.

7.1.2 EVENT MODEL

JMF uses a structured event reporting mechanism to keep JMF-based programs informed of the current state of the media system and enable JMF-based programs to respond to media-driven error conditions, such as out-of data and resource unavailable conditions. Whenever a JMF object needs to report on the current conditions, it posts a MediaEvent. MediaEvent is subclassed to identify many particular types of events. These objects follow the established Java Beans patterns for events. For each type of JMF object that can post MediaEvents, JMF defines a corresponding listener interface. To receive notification when a MediaEvent is posted, you implement the appropriate listener interface and register your listener class with the object that posts the event by calling its addListener method. Controller objects (such as Players and Processors) and certain Control objects such as GainControl post media events.

RTPSessionManager objects also post events. For more information, see RTP Events. Data Model JMF media players usually use DataSources to manage the transfer of media-content. A DataSource encapsulates both the location of media and the protocol and software used to deliver the media. Once obtained, the source

cannot be reused to deliver other media, A DataSource is identified by either a JMF MediaLocator or a URL (universal resource locator). A MediaLocator is similar to a URL and can be constructed from a URL, but can be constructed even if the corresponding protocol handler is not installed on the system. (In Java, a URL can only be constructed if the corresponding protocol handler is installed on the system.) A DataSource manages a set of SourceStream objects. A standard data source uses a byte array as the unit of transfer. A buffer data source uses a Buffer object as its unit of transfer. JMF defines several types of DataSource objects:

7.1.3 PUSH AND PULL DATA SOURCES

Media data can be obtained from a variety of sources, such as local or network files and live broadcasts. JMF data sources can be categorized according to how data transfer is initiated: Pull Data-Source--the client initiates the data transfer and controls the flow of data from pull data-sources. Established protocols for this type of data include Hypertext Transfer Protocol (HTTP) and FILE. JMF defines two types of pull data sources: PullDataSource and PullBufferDataSource, which uses a Buffer object as its unit of transfer. Push Data-Source--the server initiates the data transfer and controls the flow of data from a push data-source. Push data-sources include broadcast media, multicast media, and video-on-demand (VOD). For broadcast data, one protocol is the Real-time Transport Protocol (RTP), under development by the Internet Engineering Task Force (IETF). The MediaBase protocol developed by SGI is one protocol used for VOD. JMF defines two types of push data sources: PushDataSource and PushBufferDataSource, which uses a Buffer object as its unit of transfer.The degree of control that a client program can extend to the user depends on the

type of data source being presented. For example, an MPEG file can be repositioned and a client program could allow the user to replay the video clip or seek to a new position in the video. In contrast, broadcast media is under server control and cannot be repositioned. Some VOD protocols might support limited user control--for example, a client program might be able to allow the user to seek to a new position, but not fast forward or rewind.

7.1.4 SPECIALTY DATASOURCES

JMF defines two types of specialty data sources, cloneable data sources and merging data sources. A cloneable data source can be used to create clones of either a pull or push DataSource.To create a cloneable DataSource, you call the Manager createCloneableDataSource method and pass in the DataSource you want to clone.Once a DataSource has been passed to createCloneableDataSource, you should only interact with the cloneable DataSource and its clones; the original DataSource should no longer be used directly. Cloneable data sources implement the SourceCloneable interface, which defines one method, createClone. By calling createClone, you can create any number of clones of the DataSource that was used to construct the cloneable DataSource. The clones can be controlled through the cloneable DataSource used to create them--when connect, disconnect, start, or stop is called on the cloneable DataSource, the method calls are propagated to the clones.

The clones don't necessarily have the same properties as the cloneable data source used to create them or the original DataSource. For example, a cloneable data source created for a capture device might function as a master data source for its clones--in this case, unless the cloneable data source is used, the clones won't produce any data. If you hook up both the cloneable

data source and one or more clones, the clones will produce data at the same rate as the master. A MergingDataSource can be used to combine the SourceStreams from several DataSources into a single DataSource. This enables a set of DataSources to be managed from a single point of control--when connect, disconnect, start, or stop is called on the MergingDataSource, the method calls are propagated to the merged DataSources. To construct a MergingDataSource, you call the Manager createMergingDataSource method and pass in an array that contains the data sources you want to merge To be merged, all of the DataSources must be of the same type; for example, you cannot merge a PullDataSource and a PushDataSource. The duration of the merged DataSource is the maximum of the merged DataSource objects' durations. The ContentType is application/mixed-media. Data Formats The exact media format of an object is represented by a Format object. The format itself carries no encoding-specific parameters or global timing information, it describes the format's encoding name and the type of data the format requires. JMF extends Format to define audio- and video-specific formats. An AudioFormat describes the attributes specific to an audio format, such as sample rate, bits per sample, and number of channels. A VideoFormat encapsulates information relevant to video data. Several formats are derived from VideoFormat to describe the attributes of common video formats, including:IndexedColorFormat RGBFormat YUVFormat JPEGFormat H261Format H263Format . To receive notification of format changes from a Controller, you implement the ControllerListener interface and listen for FormatChangeEvents. (For more information, see Responding to Media Events.) Controls JMF Control provides a mechanism for setting and querying attributes of an object. A Control often provides access to a

corresponding user interface component that enables user control over an object's attributes. Many JMF objects expose Controls, including Controller objects, DataSource objects, DataSink objects, and JMF plug-ins. Any JMF object that wants to provide access to its corresponding Control objects can implement the Controls interface. Controls defines methods for retrieving associated Control objects. DataSource and PlugIn use the Controls interface to provide access to their Control objects. Standard Controls JMF defines the standard Control interfaces shown in Figure 2-8:, "JMF controls."

CachingControl enables download progress to be monitored and displayed. If a Player or Processor can report its download progress, it implements this interface so that a progress bar can be displayed to the user. GainControl enables audio volume adjustments such as setting the level and muting the output of a Player or Processor. It also supports a listener mechanism for volume changes.

DataSink or Multiplexer objects that read media from a DataSource and write it out to a destination such as a file can implement the StreamWriterControl interface. This Control enables the user to limit the size of the stream that is created. FramePositioningControl and FrameGrabbingControl export frame-based capabilities for Players and Processors. FramePositioningControl enables precise frame positioning within a Player or Processor object's media stream. FrameGrabbingControl provides a mechanism for grabbing a still video frame from the video stream. The FrameGrabbingControl can also be supported at the Renderer level. Objects that have a Format can implement the FormatControl interface to provide access to the Format. FormatControl also provides methods for querying and setting the format. A TrackControl is a type of

FormatControl that provides the mechanism for controlling what processing a Processor object performs on a particular track of media data. With the TrackControl methods, you can specify what format conversions are performed on individual tracks and select the Effect, Codec, or Renderer plug-ins that are used by the Processor. (For more information about processing media data, see Processing Time-Based Media with JMF.)

7.1.5 PLAYERS

A Player processes an input stream of media data and renders it at a precise time. A DataSource is used to deliver the input media-stream to the Player.The rendering destination depends on the type of media being presented.

A Player does not provide any control over the processing that it performs or how it renders the media data. Player supports standardized user control and relaxes some of the operational restrictions imposed by Clock and Controller.

7.1.5.1 PLAYER STATES

A Player can be in one of six states. The Clock interface defines the two primary states: Stopped and Started. To facilitate resource management, Controller breaks the Stopped state down into five standby states: Unrealized, Realizing, Realized, Prefetching, and Prefetched.

In normal operation, a Player steps through each state until it reaches the Started state: A Player in the Unrealized state has been instantiated, but does not yet know anything about its media. When a media Player is first created, it is Unrealized. When realize is called, a Player moves from the Unrealized state into the Realizing state. A Realizing Player is in the process of determining its resource requirements. During realization, a

Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources. (Exclusive-use resources are limited resources such as particular hardware devices that can only be used by one Player at a time; such resources are acquired during Prefetching.) A Realizing Player often downloads assets over the network.

When a Player finishes Realizing, it moves into the Realized state. A Realized Player knows what resources it needs and information about the type of media it is to present. Because a Realized Player knows how to render its data, it can provide visual components and controls. Its connections to oither objects in the system are in place, but it does not own any resources that would prevent another Player from starting. When prefetch is called, a Player moves from the Realized state into the Prefetching state. A Prefetching Player is preparing to present its media. During this phase, the Player preloads its media data, obtains exclusive-use resources, and does whatever else it needs to do to prepare itself to play. Prefetching might have to recur if a Player object's media presentation is repositioned, or if a change in the Player object's rate requires that additional buffers be acquired or alternate processing take place. When a Player finishes Prefetching, it moves into the Prefetched state. A Prefetched Player is ready to be started. Calling start puts a Player into the Started state. A Started Player object's time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data. A Player posts TransitionEvents as it moves from one state to another. The ControllerListener interface provides a way for your program to determine what state a Player is in and to respond appropriately. For example, when

your program calls an asynchronous method on a Player or Processor, it needs to listen for the appropriate event to determine when the operation is complete. Using this event reporting mechanism, you can manage a Player object's start latency by controlling when it begins Realizing and Prefetching. It also enables you to determine whether or not the Player is in an appropriate state before calling methods on the Player.

7.1.6 PROCESSORS

Processors can also be used to present media data. A Processor is just a specialized type of Player that provides control over what processing is performed on the input media stream. A Processor supports all of the same presentation controls as a Player.

In addition to rendering media data to presentation devices, a Processor can output media data through a DataSource so that it can be presented by another Player or Processor, further manipulated by another Processor, or delivered to some other destination, such as a file. For more information about Processors, see Processing. Presentation Controls In addition to the standard presentation controls defined by Controller, a Player or Processor might also provide a way to adjust the playback volume. If so, you can retrieve its GainControl by calling getGainControl. A GainControl object posts a GainChangeEvent whenever the gain is modified. By implementing the GainChangeListener interface, you can respond to gain changes. For example, you might want to update a custom gain control Component. Additional custom Control types might be supported by a particular Player or Processor implementation to provide other control behaviors and expose custom user interface components. You access these controls through the getControls method. For example, the

CachingControl interface extends Control to provide a mechanism for displaying a download progress bar. If a Player can report its download progress, it implements this interface. To find out if a Player supports CachingControl, you can call getControl(CachingControl) or use getControls to get a list of all the supported Controls. Standard User Interface Components A Player or Processor generally provides two standard user interface components, a visual component and a control-panel component.You can access these Components directly through the getVisualComponent and getControlPanelComponent methods.

7.1.7 PROCESSING

A Processor is a Player that takes a DataSource as input, performs some user-defined processing on the media data, and then outputs the processed media data.

A Processor can send the output data to a presentation device or to a DataSource. If the data is sent to a DataSource, that DataSource can be used as the input to another Player or Processor, or as the input to a DataSink. While the processing performed by a Player is predefined by the implementor, a Processor allows the application developer to define the type of processing that is applied to the media data. This enables the application of effects, mixing, and compositing in real-time. The processing of the media data is split into several stages:

Demultiplexing is the process of parsing the input stream. If the stream contains multiple tracks, they are extracted and output separately. For example, a QuickTime file might be demultiplexed into separate audio and video tracks. Demultiplexing is performed automatically

whenever the input stream contains multiplexed data. Pre-Processing is the process of applying effect algorithms to the tracks extracted from the input stream. Transcoding is the process of converting each track of media data from one input format to another. When a data stream is converted from a compressed type to an uncompressed type, it is generally referred to as decoding. Conversely, converting from an uncompressed type to a compressed type is referred to as encoding. Post-Processing is the process of applying effect algorithms to decoded tracks. Multiplexing is the process of interleaving the transcoded media tracks into a single output stream. For example, separate audio and video tracks might be multiplexed into a single MPEG-1 data stream. You can specify the data type of the output stream with the Processor setOutputContentDescriptor method. Rendering is the process of presenting the media to the user. The processing at each stage is performed by a separate processing component. These processing components are JMF plug-ins. If the Processor supports TrackControls, you can select which plug-ins you want to use to process a particular track. There are five types of JMF plug-ins: Demultiplexer--parses media streams such as WAV, MPEG or QuickTime. If the stream is multiplexed, the separate tracks are extracted. Effect--performs special effects processing on a track of media data. Codec--performs data encoding and decoding.

Multiplexer--combines multiple tracks of input data into a single interleaved output stream and delivers the resulting stream as an output DataSource. Renderer--processes the media data in a track and delivers it to a destination such as a screen or speaker. Processor States A Processor has two additional standby states, Configuring and Configured, which occur before the Processor enters the Realizing state..

A Processor enters the Configuring state when configure is called. While the Processor is in the Configuring state, it connects to the DataSource, demultiplexes the input stream, and accesses information about the format of the input data. The Processor moves into the Configured state when it is connected to the DataSource and data format has been determined. When the Processor reaches the Configured state, a ConfigureCompleteEvent is posted. When Realize is called, the Processor is transitioned to the Realized state. Once the Processor is Realized it is fully constructed. While a Processor is in the Configured state, getTrackControls can be called to get the TrackControl objects for the individual tracks in the media stream. These TrackControl objects enable you specify the media processing operations that you want the Processor to perform. Calling realize directly on an Unrealized Processor automatically transitions it through the Configuring and Configured states to the Realized state. When you do this, you cannot configure the processing options through the TrackControls--the default Processor settings are used. Calls to the TrackControl methods once the Processor is in the Realized state will typically fail, though some Processor implementations might support them.

7.1.7.1 METHODS AVAILABLE IN EACH PROCESSOR STATE

Since a Processor is a type of Player, the restrictions on when methods can be called on a Player also apply to Processors. Some of the Processor-specific methods also are restricted to particular states. The following table shows the restrictions that apply to a Processor. If you call a method that is illegal in the current state, the Processor throws an error or exception.

7.1.8 PROCESSING CONTROLS

You can control what processing operations the Processor performs on a track through the TrackControl for that track. You call Processor getTrackControls to get the TrackControl objects for all of the tracks in the media stream. Through a TrackControl, you can explicitly select the Effect, Codec, and Renderer plug-ins you want to use for the track. To find out what options are available, you can query the PlugInManager to find out what plug-ins are installed. To control the transcoding that's performed on a track by a particular Codec, you can get the Controls associated with the track by calling the TrackControl getControls method. This method returns the codec controls available for the track, such as BitRateControl and QualityControl. (For more information about the codec controls defined by JMF, see Controls.) If you know the output data format that you want, you can use the setFormat method to specify the Format and let the Processor choose an appropriate codec and renderer. Alternatively, you can specify the output format when the Processor is created by using a ProcessorModel.

A ProcessorModel defines the input and output requirements for a Processor. When a ProcessorModel is passed to the appropriate Manager create method, the Manager does its best to create a Processor that meets the specified requirements. Data Output The getDataOutput method returns a Processor object's output as a DataSource. This DataSource can be used as the input to another Player or Processor or as the input to a data sink. (For more information about data sinks, see Media Data Storage and Transmission.) A Processor object's output DataSource can be of any type: PushDataSource, PushBufferDataSource, PullDataSource, or

PullBufferDataSource. Not all Processor objects output data--a Processor can render the processed data instead of outputting the data to a DataSource. A Processor that renders the media data is essentially a configurable Player. Capture A multimedia capturing device can act as a source for multimedia data delivery. For example, a microphone can capture raw audio input or a digital video capture board might deliver digital video from a camera. Such capture devices are abstracted as DataSources. For example, a device that provides timely delivery of data can be represented as a PushDataSource. Any type of DataSource can be used as a capture DataSource: PushDataSource, PushBufferDataSource, PullDataSource, or PullBufferDataSource. Some devices deliver multiple data streams--for example, an audio/video conferencing board might deliver both an audio and a video stream. The corresponding DataSource can contain multiple SourceStreams that map to the data streams provided by the device. Media Data Storage and Transmission

A DataSink is used to read media data from a DataSource and render the media to some destination--generally a destination other than a presentation device. A particular DataSink might write data to a file, write data across the network, or function as an RTP broadcaster. (For more information about using a DataSink as an RTP broadcaster, see Transmitting RTP Data With a Data Sink.) Like Players, DataSink objects are constructed through the Manager using a DataSource. A DataSink can use a StreamWriterControl to provide additional control over how data is written to a file. See Writing Media Data to a File for more information about how DataSink objects are used. Storage Controls A DataSink posts a DataSinkEvent to report on its status. A DataSinkEvent can be posted with a reason code, or the DataSink can post one of the following

DataSinkEvent subtypes: DataSinkErrorEvent, which indicates that an error occurred while the DataSink was writing data. EndOfStreamEvent, which indicates that the entire stream has successfully been written.

To respond to events posted by a DataSink, you implement the DataSinkListener interface. Extensibility You can extend JMF by implementing custom plug-ins, media handlers, and data sources. Implementing Plug-Ins By implementing one of the JMF plug-in interfaces, you can directly access and manipulate the media data associated with a Processor: Implementing the Demultiplexer interface enables you to control how individual tracks are extracted from a multiplexed media stream. Implementing the Codec interface enables you to perform the processing required to decode compressed media data, convert media data from one format to another, and encode raw media data into a compressed format. Implementing the Effect interface enables you to perform custom processing on the media data. Implementing the Multiplexer interface enables you to specify how individual tracks are combined to form a single interleaved output stream for a Processor. Implementing the Renderer interface enables you to control how data is processed and rendered to an output device. Note: The JMF Plug-In API is part of the official JMF API, but JMF Players and Processors are not required to support plug-ins. Plug-ins won't work with JMF 1.0-based Players and some Processor implementations might choose not to support them. The reference implementation of JMF 2.0 provided by Sun Microsystems, Inc. and IBM Corporation fully supports the plug-in API. Custom Codec, Effect, and Renderer plug-ins are available to a Processor through the TrackControl interface. To make a plug-in available to a default Processor

or a Processor created with a ProcessorModel, you need to register it with the PlugInManager. Once you've registered your plug-in, it is included in the list of plug-ins returned by the PlugInManager getPlugInList method and can be accessed by the Manager when it constructs a Processor object. Implementing MediaHandlers and DataSources If the JMF Plug-In API doesn't provide the degree of flexibility that you need, you can directly implement several of the key JMF interfaces: Controller, Player, Processor, DataSource, and DataSink. For example, you might want to implement a high-performance Player that is optimized to present a single media format or a Controller that manages a completely different type of time-based media.

The Manager mechanism used to construct Player, Processor, DataSource, and DataSink objects enables custom implementations of these JMF interfaces to be used seamlessly with JMF. When one of the create methods is called, the Manager uses a well-defined mechanism to locate and construct the requested object. Your custom class can be selected and constructed through this mechanism once you register a unique package prefix with the PackageManager and put your class in the appropriate place in the predefined package hierarchy. MediaHandler Construction Players, Processors, and DataSinks are all types of MediaHandlers--they all read data from a DataSource. A MediaHandler is always constructed for a particular DataSource, which can be either identified explicitly or with a MediaLocator. When one of the createMediaHandler methods is called, Manager uses the content-type name obtained from the DataSource to find and create an appropriate MediaHandler object.

JMF also supports another type of MediaHandler, MediaProxy. A MediaProxy processes content from one DataSource to create another. Typically, a MediaProxy reads a text configuration file that contains all of the information needed to make a connection to a server and obtain media data. To create a Player from a MediaProxy, Manager: Constructs a DataSource for the protocol described by the MediaLocator Uses the content-type of the DataSource to construct a MediaProxy to read the configuration file. Gets a new DataSource from the MediaProxy. Uses the content-type of the new DataSource to construct a Player.

The mechanism that Manager uses to locate and instantiate an appropriate MediaHandler for a particular DataSource is basically the same for all types of MediaHandlers: Using the list of installed content package-prefixes retrieved from PackageManager, Manager generates a search list of available MediaHandler classes. Manager steps through each class in the search list until it finds a class named Handler that can be constructed and to which it can attach the DataSource.

8. REAL TIME PROTOCOL

8.1 INTRODUCTION

The real time transport protocol provides end-to-end delivery Services for data with real time characteristics, such as interactive audio and Video. Those services include payload type identification, sequence numbering, times stamping and delivering monitoring. Application typically run RTP on top Of UDP to make use of its multiplexing and checksome services, both

protocol contribute parts of the transport protocol functionally. However RTP may be used with other suitable under lying network or transport protocols.RTP supports data transfer to multiple dstination using multicast distribution if provided by the network.

Note that RTP itself does not provide any mechanism to ensure timely delivery or provide other quality of service guarantees, but relies on lower layer services to do so. It does not guarantee delivery or prevent out of order delivery, nor does it assume that the under lying network is reliable and delivers packets in sequence. The sequence number included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence number might also be used to determine the proper location of a packet, for example in video decoding, without necessary decoding packets in sequence. While RTP is primarily designed to satisfy the needs of multi participant multimedia conferences, it is not limited to that particular application. Storage of continuous data, interactive distributed simulation, active badge, and control and measurement applications may also find RTP applicable.

RTP consists of two closely linked parts:

- The real time transport protocol (RTP), to carry data that has real time properties.
- The RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The later aspect of RTCP may be sufficient for “loosely controlled” sessions, i.e., where there is no explicit membership control and set-up, but it is not necessarily intend to support all of an application’s control communication requirements.

RTP is intended to be malleable to provide the information by a particular application and will often be integrated into the application

processing rather than being implemented as a separate layer. RTP is a protocol framework that is deliberately not complete.

8.2 RTP USE SCENARIOS

The following sections describe some aspects of use of RTP. The examples were chosen to illustrate the basic operation of applications using RTP, not to limit what RTP may be used for .In these examples RTP is carried on top of IP and UDP, and follows the conventions established by the profile for audio and video.

8.2.1 SIMPLE MULTICAST AUDIO CONFERENCE

A working group of the IETF meets to discuss the latest protocol draft, using the IP multicast services of the Internet for voice communications. Through some allocation mechanism the working group chair obtains a multicast group address and pairs of ports. One port is used for audio data, and the other is used for control (RTCP) packets.

This address and port information is distributed to the intended participants. If privacy is desired the data and control packets may be encrypted in which case an encryption key must also be generated and distributed. The exact details of these allocations and distribution mechanisms are beyond the scope of RTP.

The audio conferencing application used by each conference participant sends audio data in small chunks of say 20 ms duration. Each chunk of audio data is preceded by an RTP header ;RTP header and data are in turn contained in a UDP packet. The RTP header indicates what type of audio encoding is contained in each packet so that senders can change the encoding during a conference, for example, to accommodate a new

participant that is connected through a low - bandwidth link or react to indications of network congestion.

The Internet like other packet networks occasionally loses and records packets and delays them by variable amounts of time. To cop with these impairments, RTP header contains timing information.

This timing reconstruction is performed separately for each source of RTP packets in the conference. The sequence number can also be used by the receiver how many packets are being lost.

Since members of working group leave and join during the conference, it is useful to know who is participating at any moment and how well they are receiving the audio data. For that purpose each instance of the audio application in the conference periodically multicasts a reception report plus the name of its user on the RTCP control port. The reception report tells how well the current speaker is being received and may be used to control adaptive encoding. In addition to the user name, other identifying information may be included subject to control bandwidth limits. A site sends the RTCP BYE packet when it leaves the conference.

8.2.2 AUDIO AND VIDEOCONFERENCE

If both audio and video media are used in a conference, they are transmitted as separate RTP session RTCP packets are transmitted for each medium using two different UDP port pair and multicast address. There is no direct coupling at the RTP level between the audio and video sessions, except that a user participating in both sessions should use the same distinguished name in the RTCP packets for both so that the sessions can be associated. One motivation for this separation is to allow some participants in the conference to receive only one medium if they choose. Despite the

separation, synchronized playback of the source's audio and video can be achieved using timing information carried in the RTCP packets for both sessions.

8.2.3 MIXERS AND TRANSLATORS

So far we have assumed that all sites want to receive media data in the same format. However this may not always be appropriate. Consider the case where the participants in one area are connected through a low speed link to the majority of the conference participants who enjoy high-speed network access. Instead of forcing everyone to use a lower bandwidth, reduced quality audio encoding, an RTP level really called a mixer may be placed near the low bandwidth area. This mixer resynchronizes incoming audio packets to reconstruct the constant 20 ms spacing generated by the senders, mixers these reconstructed audio streams in to a single stream, translates the audio encoding to a lower bandwidth one and forwards the lower bandwidth packet stream across the low speed link.

These packets might be unicast to a single recipient or multicast on a different address to multiple recipients. The RTP header includes a means for mixers to identify the sources that contributed to a mixed packet so that correct talker indication can be provided at the receivers. Some of the intended participants in the audio conference may be connected with high bandwidth links but might not be directly reachable via IP multicast. For example, they might be behind an application level firewall that will not let any IP packets pass. For these sites mixing may not be necessary, in which case another type of RTP level really called a translator may be used. Two translators are installed one on either side of the firewall, with the outside one funneling all multicast packets received through a secure connection to

the translator inside the firewall. The translator inside the firwewall sends them again as multicast packets to a multicast group restricted to the site's internal network. Mixers and translators mat be for a variety of purposes. An example is a video mixer that scales the images of individual peoples in separate video streams and composites them into one video stream to stimulate a group scene. Other examples of translation include the connection of a group of hosts speaking only IP/UDP to a group of hosts that understands only ST-II, or packet-by-packet encoding translation of video streams from individual sources without resynchronization or mixing.

8.3 DEFINITIONS

- ***RTP payload:*** The data transported by RTP in a packet, for example audio samples or compressed video data. The payload format and interpretation is beyond the scope of this document.

- ***RTP packet:*** A data packet consisting of the fixed RTP header, a possibly empty list of contributing sources and the payload data. Some underlying protocols may require an encapsulation of the RTP packet to be defined. Typically one packet of the underlying protocols contains a single RTP packet, but several RTP packets may be contained if permitted by the encapsulation method.
- ***RTCP packet***: A control packet consisting of a fixed header part similar to that of RTP data packets, followed by structured elements that varies depending upon the RTCP packet type. Typically, multiple RTCP packets are sent together a compound RTCP packet

in a single packet of the underlying protocol: this is enable by the length Field in the fixed header of each RTCP packet.

- ***Port:*** The “abstraction that transport protocol used to distinguish among multiple destination within a given host computer.
- ***TCP/IP protocols:*** identify ports using small positive integers.” The transport selectors used by the OSI transport layer are equivalent to ports. RTP depends upon the lower layer protocol to provide some mechanism such as ports to multiplex the RTP and RTCP packets of the session.
- ***Transport address***: The combination of a network address and port that identifies a transport level endpoint, for example an IP address and a UDP port. Packets are transmitted a source transport address to a destination transport address.
- ***RTP session:*** The association among a set of participants communicating with RTP. For each participant, a particular pair of destination transport address defines the session. The destination transport address may be common for all participants, as in the case of IP multicast, or may be different for each, as in the case of individual unicast network address plus a common port pair. In a multi cast session, each medium is carried in a separate RTP session with its own RTCP packets. The multiple RTP session are distinguished by different port number pairs and different multicast addresses. Synchronization source (SSRC): The source of a stream of RTP packet, identified by a 32 bits numeric SSRC identifier carried in the RTP header so as not to be dependent upon the network address. All packets from a synchronization source form part of the same timing and number space, so a receiver groups

packet by a synchronization source for play back. Examples of synchronization sources include the sender to stream of packets derived from a signal source such as microphone. The SSRC is a randomly chosen value meant to globally unique with a particular RTP session. If a participant generates multiple streams in RTP session, e.g. from separate video cameras, each must be identified as different SSRC.

- ***End system:*** An application that generates the content to be sent in RTP packets and/or consumes the contents of received RTP packets. An end system can act as one or more synchronization in particular RTP session, but typically only one.
- ***Mixer:*** An intermediate system that receives RTP packets from one more sources possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet. Since the timing among multiple input sources, the mixer will make timing adjustments and generates its own timing for combined stream. Thus all data packets originating from a mixer will be identified as having the mixer as their synchronization source
- ***TRANSLATOR:*** An intermediate system that forwards RTP packets with their synchronization with their source identifier interacts. Examples of translators are replicators from multicast to unicast and application level filter in firewalls.

- ***MONITOR:*** An application that receives RTCP packets sent by participants in an RTP session, in particular the reception reports, estimates the current quality. The monitor function is likely to be built into the application participating in the session, and does not

send or receive the KTP data packets. These are called third party monitors.

- ***NON_RTP-means:*** Protocols and mechanisms that may be needed in addition to RTP to provide a usable service. In particular for multimedia processing, a control application may distribute multicast addresses ann keys for encryption formats that represents formats that that do not have predefined play load type value. For simple applications electronic mail or a conference database may also be used. The specification of each of such protocols and mechanisms is outside the scope of this document.

- ***BYTE ORDER, ALIGNMENT, and TIME FORMAT:*** All integer fields are carried in network byte order that is most significant byte first. This byte order is commonly known as big endian. The transmission order is described in detail in. All header data is aligned to its natural length i.e. 16-bit fields are aligned on even Offsets, 32 bits fields aligned at offsets divisible by four, etc. Octets designated as padding have the value zero.

Wall clock time (absolute time) is represented using the time stamp format of the Network Time Protocol (NTP). The full resolution NTP timestamp is a 64-bit unsigned fixed-point number with integer part in its 32-bits and fractional part in the last. The low 16-bits of the integer

part and the high bits of the fractional part. The high 16 bits of the integer part must be determined independently.

8.4 MULTIPLEXING RTP SESSIONS

For efficient protocol processing the number of multiplexing points should be minimized as described in the integrated layer processing design principle. In RTP multiplexing is provided by the destination transport address, which define an RTP session. For example in a teleconference composed of Audio and Video media encoded separately each medium should be carried in a separate RTP Session with its own destination transport address. It is not intended that the audio and video be carried in a single RTP session and demultiplexed based on the payload type or SSRC fields. Inter leaving packets with various payload typed but using the same SSRC would introduce several problems;

1. If one payload type were switched during a session there would be no general means to identify which of the old values the new one replaced.
2. An SSRC is defined to identify a single timing and sequence number space. Inter leaving multiple payload types would require different timing spaces if the media lock rates differ would require different sequence number spaces to tell which load type suffered packet loss.
3. The RTCP sender and receiver reports can only describe one timing and sequence number space per SSRC and do not carry a payload type failed.
4. An RTP mixer would not be able to combine interleaved streams of incompatible media into one stream.
5. Carrying multiple media in one RTP session Preclude:

The Use of different network paths or network resource allocations in appropriate reception of a subset of the media if desired, if for example just audio if video would exceed the Available bandwidth and receiver implementations that use separate processes for different media, where as using separate RTP sessions permits either single or multiple process implementations.

Count of sequence number cycles. Note that different receivers within the same session will generate different extensions to the sequence number if their start times differ significantly. It is expected that reception that reception quality feedback will be useful not only for the sender but also for other receivers and third-party monitors. The sender may modify its transmissions based on the feedback; receivers can determine whether problems are local, regional or global; network managers may use profile-independent monitors that receive only the RTCP packets and not the corresponding RTP data packets to evaluate the performance of their networks for multicast distribution.

Cumulative counts are used in both the sender information and receiver report blocks so that differences may be calculated between any two reports to make measurements over both short and long time periods, and to provide resilience against the loss of a report. The difference between the last two reports received can be used to estimate the recent quality of the distribution. The NTP timestamp is included so that rates may be calculated from these differences over the interval between two reports. Since that timestamp is independent of the clock rate for the data encoding, it is possible to implement encoding- and profile-independent quality monitors. An example calculation is the packet loss rate over the interval between two reception reports. The difference in the cumulative number of packets lost

gives the number lost during that interval. The differ3ence in the extended last sequence numbers received gives the number of packets expected during the interval. The ratio of these two is the packet loss fraction over the interval. This ration should equal the fraction lost field if the two reports are consecutive, but otherwise not. The loss rate per second can be obtained by dividing the loss fraction by the difference in NTP timestamps, expressed in seconds. The number of packets received is the number of packets expected minus the number lost. The number of packets received is the number of packets expected minus the number lost. The number of packets expected may also be used to judge the statistical validity of any loss estimates. For example, 1 out of 5 packets lost has a lower significance than 200 out of 1000.

From the sender information, a third-party monitor can calculate the average payload data rate and the average packet rate over an interval without receiving the data. Taking the ratio of the two gives the average payload size. If it can be assumed that packet loss is independent of packet size, then the number of packets received by particular receiver times the average payload size (or the corresponding packet size) gives the apparent throughput available to that receiver. In addition to the cumulative counts, which allow long-term packet loss measurements using differences between reports, the fraction lost field provides a short-term measurement from a single report. This becomes more important as the size of a session scales up enough that reception state information might not be kept for all receivers or the interval between reports becomes long enough that only one report might have been received from a particular receiver.

The inter-arrival jitter field provides a second short-term measure of network congestion. Packet loss tracks persistent congestion while the jitter

measure tracks transient congestion. The jitter measure may indicate congestion before it leads to packet loss. Since the inter-arrival jitter field is only a snapshot of the jitter at the time of a report, it may be necessary to analyze a number of reports from one receiver over time or from multiple receivers, e.g., within a single network.

8.5 RTP PROFILES AND PAYLOAD FORMAT SPECIFICATION

A complete specification of RTP for a particular application will require one or more companion documents of two types described here: profiles, and payload format specifications.

RTP may be used for a variety of application with somewhat differing requirements. The flexibility to adapt to tho9se requirements is provided by allowing multiple choices in the main protocol specification, then selecting the appropriate in a separate profile document. Typically an application will operate under only one profile so there is no explicit indication of which profiles is in use. A prof9ile for audio and video applications may be found in the companion Internet-Draft draft-ieft-avt-profile for the second type of companion document is payload format specification, which defines how a particular kind of payload data, such as H.261 encoded video, should be carried in RTP. These documents are typically titled “RTP Payload Format for XYZ Audio/Video Encoding”. Payload formats may be useful under multiple profiles and may therefore be define independently of any particular profile. The profile documents are then responsible for assigning a default mapping of that format to a payload type value if needed.

Within this specification, the following items have been identified for possible definition within a profile, but this list is not meant to be exhaustive.

RTP data header: The octet in the RTP data header that contains the marker bit and payload type field may be redefined by a profile to suit different requirements, for example with more or fewer market bits.

Payload types: Assuming that a payload type field is included, the profile will usually define a set of payload formats (e.g., media encoding) and a default static mapping of those formats to payload type values. Some of the payload formats may be defined by reference to separate payload format specifications. For each payload type define, the profile must specify the RTP timestamp clock rate to be use.

RTP data header additions: Additional fields maybe appended to the fixed RTP data header if some additional functionality is required across the profile's class of applications independent of payload type

RTP data header extensions: The contents of the first 16 bits of the RTP data header extension structure must be defined if use of that mechanism is allowed under the profile for implementation-specific extensions.

RTCP report interval: A profile should specify that the values suggested for the constants employed in the calculation of the RTCP report interval will be used. Those are the RTCP fraction of session bandwidth, the minimum report interval, and the bandwidth split between senders and receivers. A profile may specify alternate values if they have been demonstrated to work in a scalable manner.

9. CLASS HIERARCHY

9.1 JCOMM

- class java.lang.Object
- interface javax.comm.CommDriver
- class javax.comm.CommPort
- class javax.comm.ParallelPort
- class javax.comm.SerialPort
- class javax.comm.CommPortIdentifier
- interface javax.comm.CommPortOwnershipListener (extends
- java.util.EventListener)
- class java.util.EventObject (implements java.io.Serializable)
- class javax.comm.ParallelPortEvent
- class javax.comm.SerialPortEvent
- interface javax.comm.ParallelPortEventListener (extends
- java.util.EventListener)
- interface javax.comm.SerialPortEventListener (extends
- java.util.EventListener)
- class java.lang.Throwable (implements java.io.Serializable)
- class java.lang.Exception
- class javax.comm.NoSuchPortException
- class javax.comm.PortInUseException
- class javax.comm.UnsupportedCommOperationException

10. CONCLUSION

This ICM server is the first server in Pakistan through which you can talk to your friend while he is surfing on Internet. The server is in final shape and working properly. We have made this server as a research project which can be extended commercially.

11. FUTURE ENHANCEMENTS

- **AUDIO ON DEMAND**

  We can make a server that can entertain the request of its users on telephone and playback any desired audio.

- **VOICE MAILING SYSTEM THROUGH TELEPHONE EXCHANGE**

  A voice mail can be recorded and sent to the desired user. The voice mail can be send in two different ways

- **PHONE TO PHONE**

  A person can send voice mail to a person's inbox, which can be listened by calling to the server from any telephone no and accessing the inbox.

- **PHONE TO PC**

  Voice or text mail can be send to a person's email account. Which can be easily accessed from the Internet.

- **ANSWERING MACHINE**

  If a person is not present at his/her place then it can be enhanced as an answering machine.

- **MULTIPLE USERS**

  This project can be enhanced for commercial purposes. Just like Internet service providers this can be made a service provider through which anyone can be benefited from this project.

  - **Soft Exchange**
  - **Phone To Phone**

12. BIBLIOGRAPHY.

Resources:

1. http://java.sun.com/products/javacomm/
2. http://www.embedded.com/98/toc9801.htm
3. http:ridgewater.mnscu.edu/classes/dc/io/
4. The book Understanding Data Communications, by Gilbert Held and George
5. http://www.clbooks.com/sqlnut/SP/
6. http:bbec.com/catalog/software/serialte.html
7. http://www.openbsd.org/
8. http://www.javaworld.com/javaworld/jw-12-1997/jw-12-javadev.html
9. The January issue of JavaWorld ran the second article in the smart card series: "Smart cards and the OpenCard Framework"
10. http://www.javaworld.com/javaworld/jw-01-1998/jw-01-javadev.html
11. In JavaWorld's February issue, you can read "Get a jumpstart on the Java Card"
12. http://www.javaworld.com/javaworld/jw-02-1998/jw-02-javadev.html
13. Also in the February issue is "Giving currency to the Java Card API"

14. http://www.javaworld.com/javaworld/jw-02-1998/jw-02-javacard.html

15. For more on Java Card 2.0, see "Understanding Java Card 2.0" in the March issue of JavaWorld

16. http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html

17. How to program JAVA by Dietel and Dietel

18. JMF Guide