VOICE MAILING SYSTEM
(VMS)

Syndicate Members

PC OMAIR ZUBAIR CHAUDHRY
PC ZUNAIR JAVED BUTT
NC UMER HASNAIN

Directing Staff (DS)

LT COL AHSEN SAEED ZAIDI

A Dissertation Submitted To

National University of Sciences & Technology
In partial fulfillment of the requirements
For the degree of

Bachelors of Engineering (BE) Computer Software

Department of Computer Science
Military College of Signals
Rawalpindi

April 2002

DECLARATION

"No portion of the work presented in this dissertation has been submitted in support of another award or qualifications either at this institution or else where."

ACKNOWLEDGEMENTS

We could not done this project all on our own. First of all we would like to thank Almighty Allah for giving us courage, showing us the way and helping us in every difficulty. We would also like to thank all our teachers from the 1st semester to the last they all were great and they given us much knowledge. We would also like to thank all our colleagues and friends and everyone who wished us luck.

I would like to thank our DS for his great help and guidance. My group members have been great thanks for sharing the workload with me. Special thanks to my parents and my two sisters. Also Mehreen for her love and support.

Omair Zubair Chaudhry

My sincere gratitude for my parents, colleagues and especially Omair and Zunair thanks for your help all the way. My special thanks are to Sadia for her continuous support through out the work.

Umer Hasnain

I'll use these few lines for mentioning all those people who have helped me in every possible way. My parents who have always been real supportive. Omair and Umer you were a great team to work with. And to one face that has inspired hope through all these years.

Zunair Javed Butt

In the end we would like to thank LT COL Ahsen Saeed Zaidi for his great help and guidance. This project would not have been possible without him. Thank you sir.

TABLE OF CONTENTS

Chapter 1

1. OVERVIEW

1.1 PURPOSE OF DOCUMENT

The document is meant to provide an insight into our project. This document contains the data of the complete development and design background of project. The purpose is to give reader a complete insight of what and how we have built the software.

1.2 SCOPE OF DOCUMENT

We have developed this document keeping in view its expected readers. We have made it as simple as possible but in doing that we have made no compromises on the technical issues of the product. In short the document is quite moderate and can be quite well understood.

1.3 PROJECT DESCRIPTION

Keeping in mind the importance and increasing users of emails and also keeping in mind that all the time we don't have a PC or a laptop with us

nor everyone has a WAP enabled mobile. So our group decided to build a software that will enable a user to access his/her email using a simple touch tone telephone from anywhere around the world.

Chapter 2

2. BACKGROUND

Before starting the project we carried out feasibility study of this project. We came across a few applications that offer similar kind of services. But they each had certain constraints like some worked more properly only in Canada and US. Whereas some required voice messaging phone for their operation. But we studied and searched a lot and we were convinced by our research that our project is quite possible.

We came to know that we might require special exchange cards for our project, which was quite expensive around about Rs.50000/-

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.

Chapter 3

3. INTRODUCTION

Nearly everyone these days have an email account. And nearly 2/3 businesses around the world are based on emails. This tells us the importance of email in our life. These days it has become a necessity for nearly everyone. But we don't have computers all the time with us or WAP mobile so how can we get access to our mails.

We can use Voice Mailing System (VMS) for this purpose.

3.1 PRODUCT PERSPECTIVE(SYSTEM MODEL)

VMS is highly interactive and simple software to use. The end user only needs to call the server which will then establish communication between the user and the server. Then it will ask the user for PINID. Then server gets the verification and then gets user emails from the email server and read the mails one by one to the user.

3.2 SYSTEM FUNCTIONALITY

VMS is a very easy and friendly usage system. Nearly everyone in the world knows how to operate a telephone so everyone can use our software. Because in our software the client doesn't has to be in front of a screen or something like that.

The main functions of the software are:

- User calls the server.
- The server establishes communication between itself and client.
- Welcome message along with usage instructions is played.
- User inputs PIN Id through touch-tones.
- Server detects the DTMF tones.
- Verifies the user
- Retrieve mails.
- Read each mail.

3.3 GENERAL CONSTRAINTS

When we started this project we had a little idea about how we are going to it. All we knew was that we have to use such a language that supports real time processing and that can integrate with other software's easily.

Keeping in view all these things we decided to implement the project using Java because it has all the features what we needed and

we all were quite good imitate server computer must have the following things to run VMS on it:

- **Software Requirements**

  JMF, JAVA COMM, JAVA MAIL APIs jar files must be in class path. IBM via Voice and 1st class mail server must be installed.

- **Hardware Requirements**

  The server needs to have voice modem installed on it.

3.4 ASSUMPTIONS AND DEPENDENCIES

The software is developed making the assumption that a caller knows how to use a simple touch-tone telephone and the caller understands English.

Chapter

4

4. SOFTWARE ARCHITECTURE

4.1 PROJECT SPECIFICATIONS

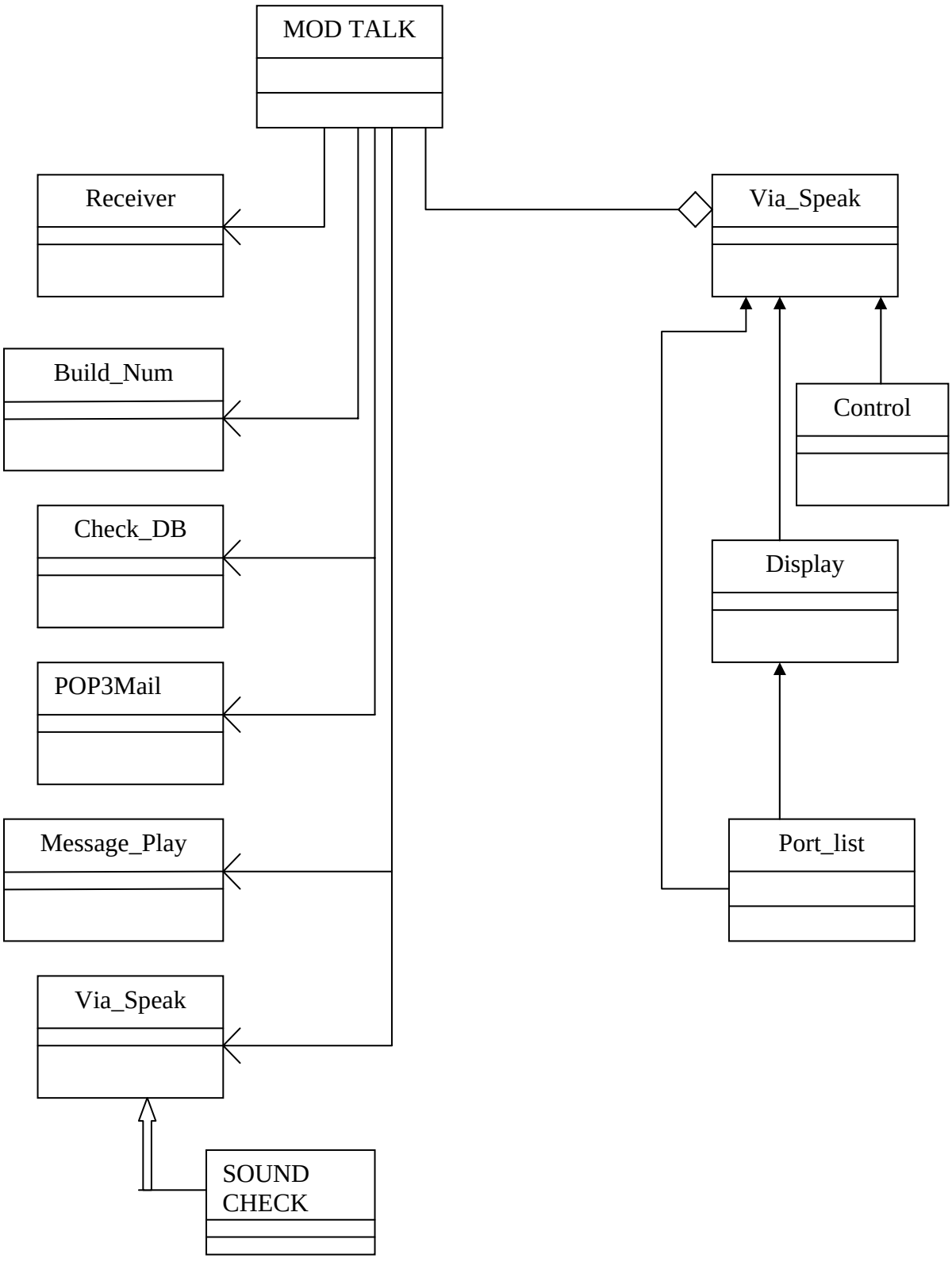
SOFTWARE REQUIREMENTS

- SOFTWARE MODULE ON THE SERVER SIDE.
- TEXT TO SPEECH CONVERTER COMPATABLE WITH JAVA SPEECH.
- MAIL SERVER COMPATABLE WITH JAVA MAIL.
- JMF
- JCOMM

HARDWARE REQUIREMENTS

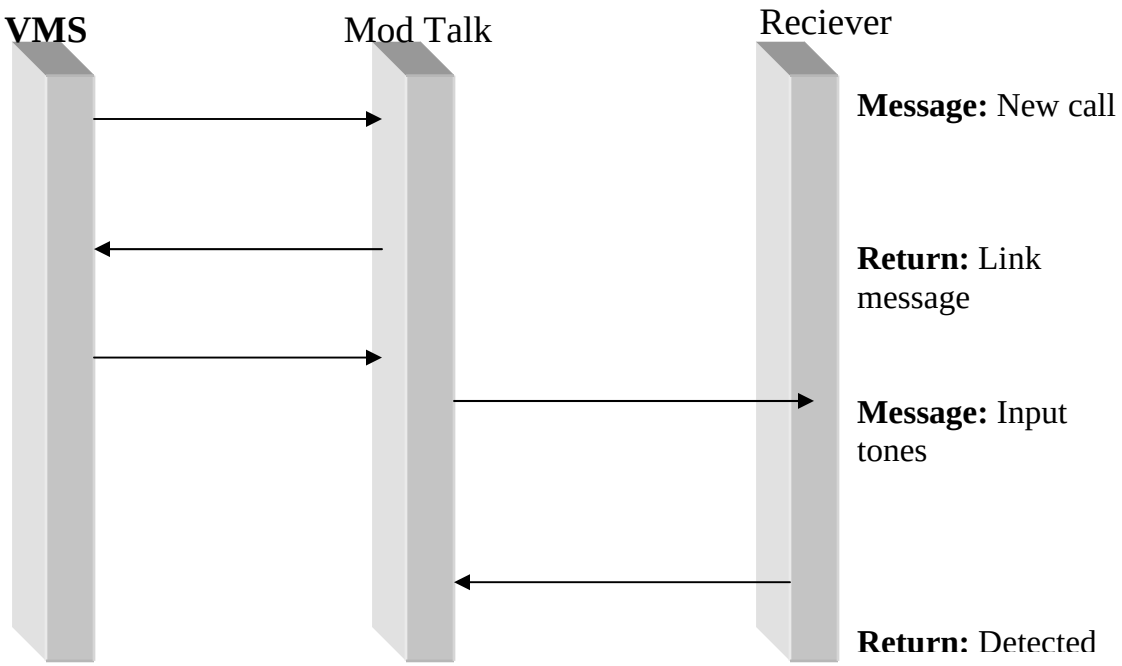
- SIMPLE TOUCH-TONE TELEPHONE
- TELEPHONE EXCHANGE CARD (FOR MULTIUSER ONLY)
- FULL DUPLEX VOICE MODEM
- TELEPHONE LINE

4.2 CLASS RELATION DIAGRAM

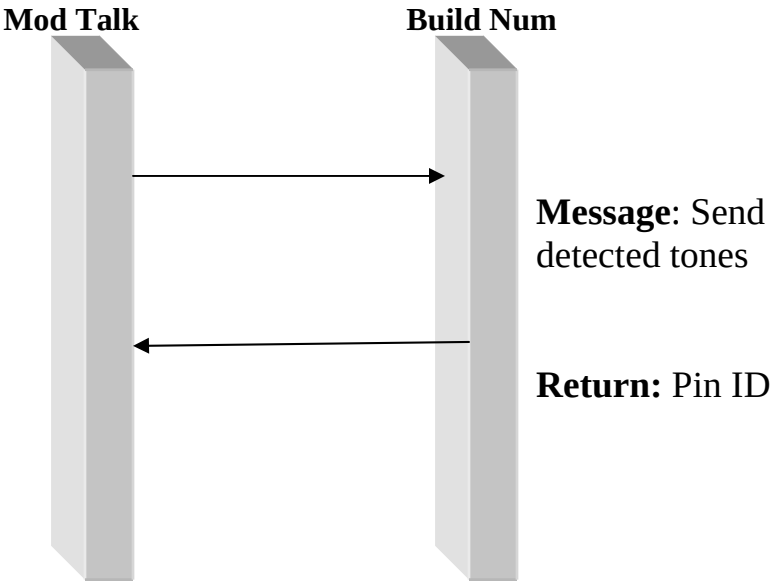


4.3 SEQUENCE DIAGRAMS

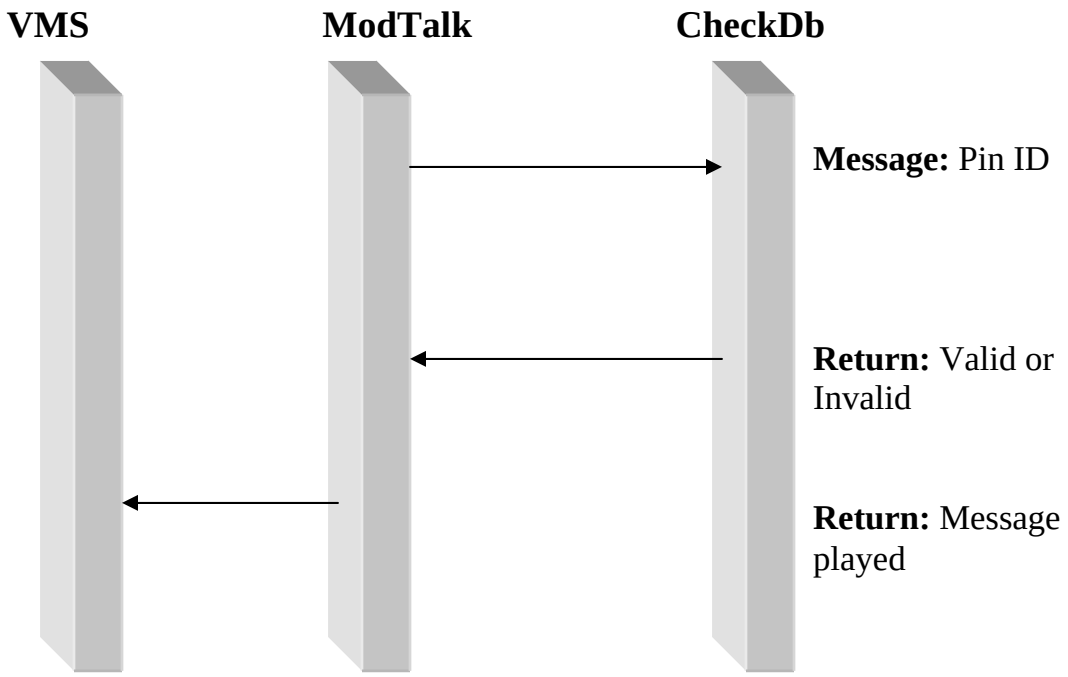
New Call:



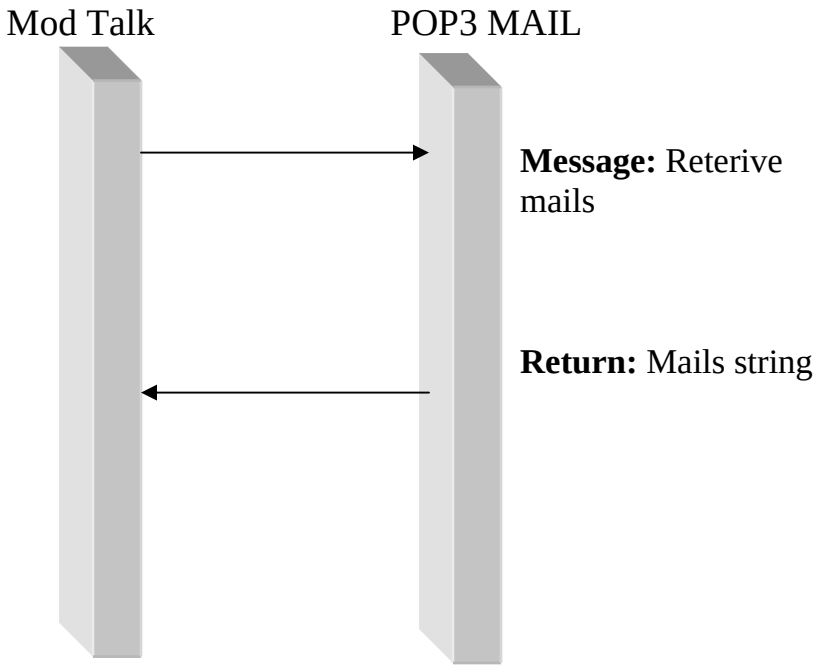
Build String:



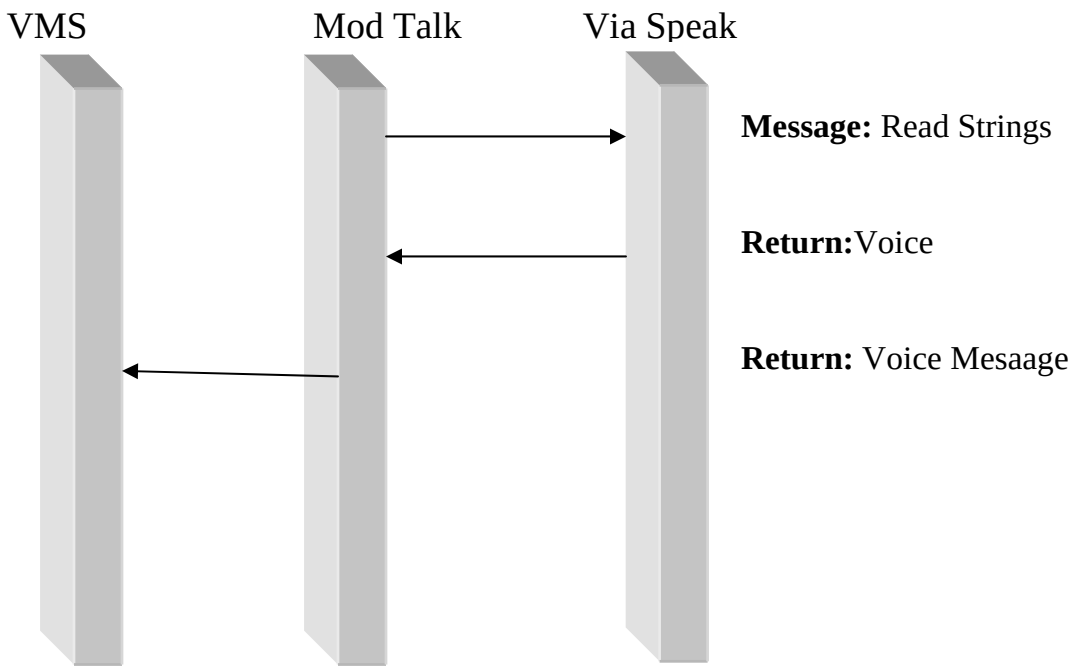
Check Pin Id :



Get Mail :



Speak :



4.4 CLASS HIERARCHY

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

- class javax.swing.JComponent (implements java.io.Serializable)
  - class javax.swing.JPanel (implements javax.accessibility.Accessible)
    - class vms.**PanelPaint**
- 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 vms.**VmsGUI** (implements java.awt.event.ActionListener, java.lang.Runnable)

- class vms.**MsgPlayer** (implements javax.media.ControllerListener)
- class vms.**Pop3Mail**
- class vms.**PortLister**
- class vms.**SndTest**
- class java.lang.Thread (implements java.lang.Runnable)
  - class vms.**ModTalk**
  - class vms.**ModTalk.Responder**
- class vms.**ViaSpeak**
- class vms.**ViaSpeak.MyListener** (implements javax.speech.synthesis.SpeakableListener)

4.5 LIST OF ALL MEMBER FUNCTIONS

A

actionPerformed(ActionEvent) - Method in class vms.VmsGUI

addComp(Component, int, int, int, int) - Method in class vms.VmsGUI
Method used internally for adding GUI components to frame.
alloc() - Method in class vms.ViaSpeak
This method allocates the resources when it is called from the constructor.

B

bubs() - Method in class vms.ModTalk
Final clean up method.
buildID(char) - Method in class vms.BuildNum
The method takes as input a character and appends it to a string.
BuildNum - class vms.BuildNum.
Title: Class BuildNum Description: This class collects the character passed to it and returns a fully built String.
BuildNum(int) - Constructor for class vms.BuildNum
The constructor takes an int as input which is the maximum length of the string.

C

CheckDB - class vms.CheckDB.
Title: Class CheckDB Description: This class takes a user ID and pin code and verifies it against a database.
CheckDB(String, String) - Constructor for class vms.CheckDB
The input to this constructor is a user id and a pin code.
checkMail() - Method in class vms.Pop3Mail
This method is called from the contructor to carry out the e-mail retrieval operation.
cleanUp() - Method in class vms.ModTalk
This is an intermediate method to release soem of the resources of the system.
collector(BuildNum) - Method in class vms.ModTalk

This method collects the DTMF tone digits punched by the user and returns a complete string.
controllerUpdate(ControllerEvent) - Method in class vms.MsgPlayer
This method listens for different events of the player.

D

deAlloc() - Method in class vms.ViaSpeak
This method de-allocates the resources claimed.
driver() - Method in class vms.ModTalk
This method is basically the starting point of all the activities of the class.

G

getCountme() - Method in class vms.CheckDB

getDriver() - Method in class vms.CheckDB

getPassword() - Method in class vms.CheckDB

getRecnum() - Method in class vms.CheckDB

getResultsVector() - Method in class vms.CheckDB

getSql() - Method in class vms.CheckDB

getUrl() - Method in class vms.CheckDB

getUsername() - Method in class vms.CheckDB

I

initializer() - Method in class vms.ModTalk
This method initializes various objects for the system.

M

mailReader(String[]) - Method in class vms.ModTalk
It calls the Pop3mail class for retrieving mail of the client/user.
main(String[]) - Static method in class vms.VmsGUI
Main method for starting the VMS system.
makeCn() - Method in class vms.CheckDB
This method makes a connection with the database.
markerReached(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

ModTalk - class vms.ModTalk.
Title: Class ModTalk Description: This class basically handles the Modem comm.
ModTalk.Responder - class vms.ModTalk.Responder.
This internal class accomplishes the task of listening on the modem for the response() method of class ModTalk
ModTalk.Responder(ModTalk) - Constructor for class vms.ModTalk.Responder

ModTalk() - Constructor for class vms.ModTalk
This constructor is empty.
ModTalk(String, int) - Constructor for class vms.ModTalk
This constructor takes as input the port name at which the modem is installed and the max number of retries.
MsgPlayer - class vms.MsgPlayer.
Title: Class MsgPlayer Description: This class is used to play pre-recorded mesages for the sytem.
MsgPlayer(String) - Constructor for class vms.MsgPlayer
This constructor takes as input the name of the sound file to be played.
msgProvider() - Method in class vms.ModTalk
This method makes available system messages which may be used by a GUI built for the system.
msgSetter(String) - Method in class vms.ModTalk
It sets the currently generated system message.

P

paintComponent(Graphics) - Method in class vms.PanelPaint
Method that actually paints the image.

PanelPaint - class vms.PanelPaint.
Title: Class PanelPaint Description: This class is simply paints the images passed to it on a panel.
PanelPaint(ImageIcon) - Constructor for class vms.PanelPaint
The input is an ImageIcon object which is to be painted on a panel.
Pop3Mail - class vms.Pop3Mail.
Title: Class Pop3Mail Description: This class retrieves the e-mails from a pop3 mail server.
Pop3Mail() - Constructor for class vms.Pop3Mail
This constructor is only for testing.
Pop3Mail(String, String, String) - Constructor for class vms.Pop3Mail
This constructor takes as input the username,password and address of pop3 mail server from which to e-mails are to be retrieved.
PortLister - class vms.PortLister.
Title: Class PortLister Description: This class simply enumerates the Com ports installed on the system.
PortLister() - Constructor for class vms.PortLister

R

response() - Method in class vms.ModTalk
This method listens on the modem for response to the AT commands or other events.
run() - Method in class vms.VmsGUI

run() - Method in class vms.ModTalk

run() - Method in class vms.ModTalk.Responder

runCmd(String) - Method in class vms.ModTalk
This method takes as input an AT command to run on the modem.

S

setDriver(String) - Method in class vms.CheckDB

setPassword(String) - Method in class vms.CheckDB

setSql(String) - Method in class vms.CheckDB

setUrl(String) - Method in class vms.CheckDB

setUsername(String) - Method in class vms.CheckDB

shutdown() - Method in class vms.ModTalk
It is usually called at shutdown of the system for completly releasing all claimed resources.
SndTest - class vms.SndTest.
Title: Class SndTest Description: This class puts the sound played on the modem mixer.
SndTest() - Constructor for class vms.SndTest

speakableCancelled(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

speakableEnded(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

speakablePaused(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

speakableResumed(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

speakableStarted(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

speakMe(String) - Method in class vms.ViaSpeak
This method passes on to theTTS engine the string to be speech synthesized.
startVms() - Method in class vms.VmsGUI
This method starts the VMS system.
stopVms() - Method in class vms.VmsGUI
This method stops the VMS system.
synStop() - Method in class vms.ViaSpeak
This method stops the current speech synthesizer.
synStopAll() - Method in class vms.ViaSpeak
This method stops all the queued up speech synthesizers.

T

takeEntry() - Method in class vms.ModTalk
This method handles the interaction with the client/user.
topOfQueue(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

tryAgain() - Method in class vms.ModTalk
It allows the user to re-attempt identification with the system.

V

ViaSpeak - class vms.ViaSpeak.
Title: Class ViaSpeak Description: This class is the implemetation of text to speech converter.
ViaSpeak.MyListener - class vms.ViaSpeak.MyListener.
This internal class listens for different events on the TTS engine.
ViaSpeak.MyListener(ViaSpeak) - Constructor for class vms.ViaSpeak.MyListener

ViaSpeak() - Constructor for class vms.ViaSpeak
The construcutor allocates the resources.
vms - package vms

VmsGUI - class vms.VmsGUI.
Title: Class VmsGUI Description: This class provides the GUI for the server side handling of the VMS system.
VmsGUI() - Constructor for class vms.VmsGUI

W

wordStarted(SpeakableEvent) - Method in class vms.ViaSpeak.MyListener

---

4.6 ALL CLASSES

4.6.1 Class VmsGUI

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

All Implemented Interfaces:
javax.accessibility.Accessible, java.awt.event.ActionListener, java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer, javax.swing.RootPaneContainer, java.lang.Runnable, java.io.Serializable, javax.swing.WindowConstants

---

public class **VmsGUI**
extends javax.swing.JFrame
implements java.awt.event.ActionListener, java.lang.Runnable
Title: Class VmsGUI Description: This class provides the GUI for the server side handling of the VMS system.

See Also:
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	
VmsGUI()	

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent e)
void	**addComp**(java.awt.Component comp, int row, int col, int w, int h) Method used internally for adding GUI components to frame.
static void	**main**(java.lang.String[] args) Main method for starting the VMS system.
void	**run**()
void	**startVms**() This method starts the VMS system.
void	**stopVms**() This method stops the VMS system.

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

VmsGUI
public **VmsGUI**()

Method Detail

run
public void **run**()

Specified by:

run in interface java.lang.Runnable

---

actionPerformed
public void **actionPerformed**(java.awt.event.ActionEvent e)

Specified by:

actionPerformed in interface java.awt.event.ActionListener

---

startVms
public void **startVms**()
This method starts the VMS system. It should not be called directly.

---

stopVms
public void **stopVms**()
This method stops the VMS system. It should not be called directly.

---

addComp
public void **addComp**(java.awt.Component comp,
int row,
int col,
int w,
int h)

Method used internally for adding GUI components to frame. It should not be called directly.

---

main
public static void **main**(java.lang.String[] args)
Main method for starting the VMS system.

4.6.2 Class ModTalk

java.lang.Object
|
+--java.lang.Thread
|
+--**vms.ModTalk**

All Implemented Interfaces:
java.lang.Runnable

---

public class **ModTalk**
extends java.lang.Thread
Title: Class ModTalk Description: This class basically handles the Modem comm. and coordinates the activities for all other classes. This class is the core of the system.

Inner Class Summary	
class	**ModTalk.Responder** This internal class accomplishes the task of listening on the modem for the response() method of class ModTalk

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

ModTalk() This constructor is empty.
ModTalk(java.lang.String portName, int numtries) This constructor takes as input the port name at which the modem is installed and the max number of retries.

Method Summary	
void	**bubs**() Final clean up method.
void	**cleanUp**() This is an intermediate method to release soem of the resources of the system.
java.lang.String	**collector**(BuildNum bnum) This method collects the DTMF tone digits punched by the user and returns a complete string.
void	**driver**() This method is basically the starting point of all the activities of the class.
void	**initializer**() This method initializes various objects for the system.
void	**mailReader**(java.lang.String[] mailBox) It calls the Pop3mail class for retrieving mail of the client/user.
java.lang.String	**msgProvider**() This method makes available system messages which may be used by a GUI built for the system.
void	**msgSetter**(java.lang.String sysMsg) It sets the currently generated system message.
java.lang.String	**response**() This method listens on the modem for response to

	the AT commands or other events.
void	**run**()
void	**runCmd**(java.lang.String atCommand) This method takes as input an AT command to run on the modem.
void	**shutdown**() It is usually called at shutdown of the system for completly releasing all claimed resources.
void	**takeEntry**() This method handles the interaction with the client/user.
void	**tryAgain**() It allows the user to re-attempt identification with the system.

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

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

Constructor Detail

ModTalk

public **ModTalk**()

This constructor is empty. It does nothing but create a null object which may be needed for different reasons.

ModTalk

public **ModTalk**(java.lang.String portName,
int numtries)

This constructor takes as input the port name at which the modem is installed and the max number of retries.

Method Detail

initializer

public void **initializer**()

This method initializes various objects for the system. e.g. modem, setting up the comm. port etc. It should not be called directly.

---

bubs

public void **bubs**()

Final clean up method. It releases all resources claimed by the system.

---

runCmd

public void **runCmd**(java.lang.String atCommand)

This method takes as input an AT command to run on the modem.

---

response

public java.lang.String **response**()

This method listens on the modem for response to the AT commands or other events.

---

driver

public void **driver**()

This method is basically the starting point of all the activities of the class.

---

takeEntry

public void **takeEntry**()

This method handles the interaction with the client/user. e.g. taking input of the user id and pin code.

---

mailReader

public void **mailReader**(java.lang.String[] mailBox)

It calls the Pop3mail class for retrieving mail of the client/user.

---

tryAgain

public void **tryAgain**()

It allows the user to re-attempt identification with the system. Also it keeps track of the the number of retries.

---

cleanUp

public void **cleanUp**()

This is an intermediate method to release soem of the resources of the system.

---

collector

public java.lang.String **collector**(BuildNum bnum)

This method collects the DTMF tone digits punched by the user and returns a complete string.

---

shutdown

public void **shutdown**()

It is usually called at shutdown of the system for completly releasing all claimed resources.

---

run

public void **run**()

Overrides:

run in class java.lang.Thread

---

msgSetter

public void **msgSetter**(java.lang.String sysMsg)

It sets the currently generated system message.

---

msgProvider

public java.lang.String **msgProvider**()

This method makes available system messages, which may be used by a GUI built for the system.

4.6.3 Class MsgPlayer

```
java.lang.Object
  |
  +--vms.MsgPlayer
```

All Implemented Interfaces:

javax.media.ControllerListener

---

public class **MsgPlayer**
extends java.lang.Object
implements javax.media.ControllerListener

Title: Class MsgPlayer Description: This class is used to play pre-recorded mesages for the sytem.

Constructor Summary
MsgPlayer(java.lang.String msgFile) This constructor takes as input the name of the sound file to be played.

Method Summary	
void	**controllerUpdate**(javax.media.ControllerEvent event) This method listens for different events of the player.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MsgPlayer

public MsgPlayer(java.lang.String msgFile)

This constructor takes as input the name of the sound file to be played.

Method Detail

controllerUpdate

public void controllerUpdate(javax.media.ControllerEvent event)

This method listens for different events of the player.

Specified by:

controllerUpdate in interface javax.media.ControllerListener

4.6.4 Class PortLister

java.lang.Object
|
+--**vms.PortLister**

---

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

Title: Class PortLister Description: This class simply enumerates the Com ports installed on the system.

Constructor Summary	
PortLister()	

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

Constructor Detail

PortLister

public PortLister()

4.6.5 Class PanelPaint

```
java.lang.Object
  |
  +--java.awt.Component
       |
       +--java.awt.Container
            |
            +--javax.swing.JComponent
                 |
                 +--javax.swing.JPanel
                      |
                      +--vms.PanelPaint
```

All Implemented Interfaces:

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

---

public class **PanelPaint**
extends javax.swing.JPanel

Title: Class PanelPaint Description: This class is simply paints the images passed to it on a panel. This image painted panel may be used to display on a GUI.

See Also:

Serialized Form

Inner classes inherited from class javax.swing.JPanel
javax.swing.JPanel.AccessibleJPanel

Inner classes inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

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.JComponent
accessibleContext, listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

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	
PanelPaint(javax.swing.ImageIcon ic) The input is an ImageIcon object which is to be painted on a panel.	

Method Summary	
void	**paintComponent**(java.awt.Graphics g) Method that actually paints the image.

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUIClassID, paramString, updateUI

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addPropertyChangeListener, addPropertyChangeListener, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, fireVetoableChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAutoscrolls, getBorder, getBounds, getClientProperty, getComponentGraphics, getConditionForKeyStroke, getDebugGraphicsOptions, getGraphics, getHeight, getInputMap, getInputMap, getInputVerifier, getInsets,

getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getVerifyInputWhenFocusTarget, getVisibleRect, getWidth, getX, getY, grabFocus, hasFocus, hide, isDoubleBuffered, isFocusCycleRoot, isFocusTraversable, isLightweightComponent, isManagingFocus, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isOptimizedDrawingEnabled, isPaintingTile, isPreferredSizeSet, isRequestFocusEnabled, isValidateRoot, paint, paintBorder, paintChildren, paintImmediately, paintImmediately, print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent, processFocusEvent, processKeyBinding, processKeyEvent, processMouseMotionEvent, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removePropertyChangeListener, removePropertyChangeListener, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setDebugGraphicsOptions, setDoubleBuffered, setEnabled, setFont, setForeground, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setUI, setVerifyInputWhenFocusTarget, setVisible,

unregisterKeyboardAction, update

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getLayout, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, setLayout, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, bounds, checkImage, checkImage, coalesceEvents, contains, createImage, createImage, disableEvents, dispatchEvent, enable, enableEvents, enableInputMethods, getBackground, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getInputContext, getInputMethodRequests, getLocale, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, imageUpdate, inside, isDisplayable, isEnabled, isLightweight, isShowing, isValid,

isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, processComponentEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processMouseEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus

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

Constructor Detail

PanelPaint

public PanelPaint(javax.swing.ImageIcon ic)

The input is an ImageIcon object which is to be painted on a panel.

Method Detail

paintComponent

public void paintComponent(java.awt.Graphics g)

Method that actually paints the image.

Overrides:

paintComponent in class javax.swing.JComponent

4.6.6 Class CheckDB

java.lang.Object
|
+--**vms.CheckDB**

---

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

Title: Class CheckDB Description: This class takes a user ID and pin code and verifies it against a database. A user name and a password with the name of a pop mail server is returned.

Constructor Summary	
CheckDB(java.lang.String u, java.lang.String p) The input to this constructor is a user id and a pin code.	

Method Summary	
int	**getCountme**()
java.lang.String	**getDriver**()
java.lang.String	**getPassword**()
int	**getRecnum**()
java.util.Vector	**getResultsVector**()

java.lang.String	**getSql**()
java.lang.String	**getUrl**()
java.lang.String	**getUsername**()
void	**makeCn**() This method makes a connection with the database.
void	**setDriver**(java.lang.String newDriver)
void	**setPassword**(java.lang.String newPassword)
void	**setSql**(java.lang.String newSql)
void	**setUrl**(java.lang.String newUrl)
void	**setUsername**(java.lang.String newUsername)

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

Constructor Detail

CheckDB

public CheckDB(java.lang.String u,
java.lang.String p)

The input to this constructor is a user id and a pin code.

Method Detail

makeCn

public void makeCn()

This method makes a connection with the database.

---

setUsername

public void setUsername(java.lang.String newUsername)

---

getUsername

public java.lang.String getUsername()

---

setPassword

public void setPassword(java.lang.String newPassword)

---

getPassword

public java.lang.String getPassword()

---

setUrl

public void setUrl(java.lang.String newUrl)

---

getUrl

public java.lang.String getUrl()

---

setDriver

public void setDriver(java.lang.String newDriver)

---

getDriver

public java.lang.String getDriver()

---

setSql

public void setSql(java.lang.String newSql)

---

getSql

public java.lang.String **getSql**()

---

getResultsVector

public java.util.Vector **getResultsVector**()

---

getCountme

public int **getCountme**()

---

getRecnum

public int **getRecnum**()

4.6.7 Class Pop3Mail

```
java.lang.Object
  |
  +--vms.Pop3Mail
```

---

public class **Pop3Mail**
extends java.lang.Object

Title: Class Pop3Mail Description: This class retrieves the e-mails from a pop3 mail server.

Constructor Summary
Pop3Mail() This constructor is only for testing.
Pop3Mail(java.lang.String luser, java.lang.String lpwd, java.lang.String lhost) This constructor takes as input the username,password and address of pop3 mail server from which to e-mails are to be retrieved.

Method Summary	
void	**checkMail**() This method is called from the contructor to carry out the e-mail retrieval operation.

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

Constructor Detail

Pop3Mail

public Pop3Mail()

This constructor is only for testing.

Pop3Mail

public Pop3Mail(java.lang.String luser,
java.lang.String lpwd,
java.lang.String lhost)

This constructor takes as input the username,password and address of pop3 mail server from which to e-mails are to be retrieved.

Method Detail

checkMail

public void checkMail()

This method is called from the contructor to carry out the e-mail retrieval operation.

4.6.8 Class SndTest

java.lang.Object
|
+--**vms.SndTest**

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

Title: Class SndTest Description: This class puts the sound played on the modem mixer.

Constructor Summary	
SndTest()	

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

Constructor Detail

SndTest

public SndTest()

4.6.9 Class ViaSpeak

```
java.lang.Object
  |
  +--vms.ViaSpeak
```

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

Title: Class ViaSpeak Description: This class is the implemetation of text to speech converter.

Inner Class Summary	
class	**ViaSpeak.MyListener** This internal class listens for different events on the TTS engine.

Constructor Summary	
ViaSpeak() The construcutor allocates the resources.	

Method Summary	
void	**alloc**() This method allocates the resources when it is called from the constructor.
void	**deAlloc**() This method de-allocates the resources claimed.
void	**speakMe**(java.lang.String txtRead) This method passes on to theTTS engine the string to be speech synthesized.
void	**synStop**() This method stops the current speech synthesizer.
void	**synStopAll**() This method stops all the queued up speech synthesizers.

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

Constructor Detail

ViaSpeak

public ViaSpeak()

The construcutor allocates the resources.

Method Detail

alloc

public void alloc()

This method allocates the resources when it is called from the constructor.

---

deAlloc

public void deAlloc()

This method de-allocates the resources claimed.

---

synStop

public void synStop()

This method stops the current speech synthesizer.

---

synStopAll

public void synStopAll()

This method stops all the queued up speech synthesizers.

---

speakMe

public void speakMe(java.lang.String txtRead)

This method passes on to theTTS engine the string to be speech synthesized.

4.6.10 Class BuildNum

java.lang.Object
|
+--**vms.BuildNum**

---

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

Title: Class BuildNum Description: This class collects the character passed to it and returns a fully built String.

Constructor Summary
BuildNum(int limit) The constructor takes an int as input which is the maximum length of the string.

Method Summary	
void	**buildID**(char ch) The method takes as input a character and appends it to a string.

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

Constructor Detail

BuildNum

public BuildNum(int limit)

The constructor takes an int as input which is the maximum length of the string. The string must begin and end with a '*'. A '#' resets the String to null.

Method Detail

buildID

public void buildID(char ch)

The method takes as input a character and appends it to a string

Chapter 5

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 telephones 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 commands, 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 AT 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 Ring back goes away timer (originate).

#VRN Ring back 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 ring back 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.

Chapter 6

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. Application programmers should not use them.

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.

6.3 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, and 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 diagrams 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 is 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.

6.4 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 writes 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();
} ...
```

6.5 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.

Chapter 7

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 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.

7.1.1.1 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 objects 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.

7.1.2 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. A particular Player or Processor implementation to provide other control behaviors and expose custom user interface components might support additional custom Control types. 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.

7.1.3 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 implementor predefines the processing performed by a Player, 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. This TrackControl objects enable you specify the media processing operations that you want the Processor to perform. Calling realizes 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.

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.

Chapter 8

8. TEXT TO SPEECH CONVERSION (USING JAVA)

8.1 IMPORTANCE OF SPEECH TECHNOLOGY

Speech technology is becoming increasingly important in both personal and enterprise computing as it is used to improve existing user interfaces and to support new means of human interaction with computers. Speech technology allows hands-free use of computers and supports access to computing capabilities away from the desk and over the telephone. Speech recognition and speech synthesis can improve computer accessibility for users with disabilities and can reduce the risk of repetitive strain injury and other problems caused by current interfaces.

The following sections describe some current and emerging uses of speech technology. The lists of uses are far from exhaustive. New speech products are being introduced on a weekly basis and speech technology is rapidly entering new technical domains and new markets. The coming years should see speech input and output truly revolutionize the way people interact with computers and present new and unforeseen uses of speech technology.

8.1.1 DESKTOP

Speech technology can augment traditional graphical user interfaces. At its simplest, it can be used to provide audible prompts with spoken *"Yes/No/OK"* responses that do not distract the user's focus. But increasingly, complex commands are enabling rapid access to features that have traditionally been buried in sub-menus and dialogs. For example, the command *"Use 12-point, bold, Helvetica font"* replaces multiple menu selections and mouse clicks.

Drawing, CAD and other hands-busy applications can be enhanced by using speech commands in combination with mouse and keyboard actions to improve the speed at which users can manipulate objects. For example, while dragging an object, a speech command could be used to change its color and line type all without moving the pointer to the menu-bar or a tool palette.

Natural language commands can provide improvements in efficiency but are increasingly being used in desktop environments to enhance usability. For many users it's easier and more natural to produce spoken commands than to remember the location of functions in menus and dialog boxes. Speech technology is unlikely to make existing user interfaces redundant any time soon, but spoken commands provide an elegant complement to existing interfaces.

8.1.2 TELEPHONY SYSTEMS

Speech technology is being used by many enterprises to handle customer calls and internal requests for access to information, resources and services. Speech recognition over the telephone provides a more natural and substantially more efficient interface than touch-tone systems. For example, speech recognition can "flatten out" the deep menu structures used in touch-tone systems.

Systems are already available for telephone access to email calendars and other computing facilities that have previously been available only on the desktop or with special equipment. Such systems allow convenient computer access by telephones in hotels, airports and airplanes.

8.1.3 PERSONAL AND EMBEDDED DEVICES

Speech technology is being integrated into a range of small-scale and embedded computing devices to enhance their usability and reduce their size. Such devices include Personal Digital Assistants (PDAs), telephone handsets, toys and consumer product controllers.

Speech technology is particularly compelling for such devices and is being used increasingly as the computer power of these device increases. Speech recognition through a microphone can replace input through a much larger keyboard. A speaker for speech synthesis output is also smaller than most graphical displays.

8.2 OVERVIEW OF JAVA SPEECH API

8.2.1 JAVA SPEECH API

The Java Speech API defines a standard, easy-to-use, cross-platform software interface to state-of-the-art speech technology. Two core speech technologies are supported through the Java Speech API: *speech recognition* and *speech synthesis*. Speech recognition provides computers with the ability to listen to spoken language and to determine what has been said. In other words, it processes audio input containing speech by converting it to text. Speech synthesis provides the reverse process of producing synthetic speech from text generated by an application, an applet or a user. It is often referred to as *text-to-speech* technology.

Enterprises and individuals can benefit from a wide range of applications of speech technology using the Java Speech API. For instance, interactive voice response systems are an attractive alternative to touch-tone interfaces over the telephone; dictation systems can be considerably faster than typed input for many users; speech technology improves accessibility to computers for many people with physical limitations.

Speech interfaces give Java application developers the opportunity to implement distinct and engaging personalities for their applications and to differentiate their products. Java application developers will have access to state- of-the-art speech technology from leading speech companies. With a standard API for speech, users can choose the speech products that best meet their needs and their budget.

8.2.2 SPEECH-ENABLED JAVA APPLICATIONS

The existing capabilities of the Java platform make it attractive for the development of a wide range of applications. With the addition of the Java Speech API, Java application developers can extend and complement existing user interfaces with speech input and output. For existing developers of speech applications, the Java platform now offers an attractive alternative with:

Portability*:* the Java programming language, APIs and virtual machine are available for a wide variety of hardware platforms and operating systems and are supported by major web browsers.

Powerful and compact environment*:* the Java platform provides developers with a powerful, object-oriented, garbage collected language, which enables rapid development and improved reliability.

Network aware and secure*:* from its inception, the Java platform has been network aware and has included robust security.

8.2.3 REQUIREMENTS

To use the Java Speech API, a user must have certain minimum software and hardware available. The following is a broad sample of requirements. The individual requirements of speech synthesizers and

speech recognizers can vary greatly and users should check product requirements closely.

Speech software: A JSAPI-compliant speech recognizer or synthesizer is required.

System requirements*:* most desktop speech recognizers and some speech synthesizers require relatively powerful computers to run effectively. Check the minimum and recommended requirements for CPU, memory and disk space when purchasing a speech product.

Audio Hardware: Speech synthesizers require audio output. Speech recognizers require audio input. Most desktop and laptop computers now sold have satisfactory audio support. Most dictation systems perform better with good quality sound cards.

Microphone: Desktop speech recognition systems get audio input through a microphone. Some recognizers, especially dictation systems, are sensitive to the microphone and most recognition products recommend particular microphones. Headset microphones usually provide best performance, especially in noisy environments. Tabletop microphones can be used in some environments for some applications.

8.3 SPEECH ENGINES (JAVAX.SPEECH)

8.3.1 SPEECH ENGINE

The javax.speech package of the Java Speech API defines an abstract software representation of a speech engine. "Speech engine" is the generic term for a system designed to deal with either speech input or speech output. Speech synthesizers and speech recognizers are both speech engine instances. Speaker verification systems and speaker identification systems are also speech engines but are not currently supported through the Java Speech API.

The javax.speech package defines classes and interfaces that define the basic functionality of an engine. The javax.speech.synthesis package and javax.speech.recognition package extends and augments the basic functionality to define the specific capabilities of speech synthesizers and speech recognizers.

The Java Speech API makes only one assumption about the implementation of a JSAPI engine: that it provides a true implementation of the Java classes and interfaces defined by the API. In supporting those classes and interfaces, an engine may completely software-based or may be a combination of software and hardware. The engine may be local to the client computer or remotely operating on a server. The engine may be written entirely as Java software or may be a combination of Java software and native code.

The basic processes for using a speech engine in an application are as follows.

- Identify the application's functional requirements for an engine (e.g, language or dictation capability).
- Locate and create an engine that meets those functional requirements.
- Allocate the resources for the engine.
- Set up the engine.
- Begin operation of the engine - technically, resume it.
- Use the engine
- Deallocate the resources of the engine.

8.3.2 SPEAKING TEXT :

The Synthesizer interface provides four methods for submitting text to a speech synthesizer to be spoken. These methods differ according to the formatting of the provided text, and according to the type of object from which the text is produced. All methods share one feature; they all allow a listener to be passed that will receive notifications as output of the text proceeds.

The simplest method - speakPlainText - takes text as a String object. This method is illustrated in the *"Hello World!"* example at the beginning of this chapter. As the method name implies, this method treats the input text as plain text without any of the formatting described below.

The remaining three speaking methods - all named speak - treat the input text as being specially formatted with the Java Speech Markup Language (JSML). JSML is an application of XML (eXtensible Markup

Language), a data format for structured document interchange on the internet. JSML allows application developers to annotate text with structural and presentation information to improve the speech output quality. JSML is defined in detail in a separate technical document, *"The Java Speech Markup Language Specification."*

The three speak methods retrieve the JSML text from different Java objects. The three methods are:

- void speak(Speakable text, SpeakableListener listener);
- void speak(URL text, SpeakableListener listener);
- void speak(String text, SpeakableListener listener);

8.3.3 SPEECH OUTPUT QUEUE

Each call to the speak and speakPlainText methods places an object onto the synthesizer's *speech output queue*. The speech output queue is a FIFO queue: first-in-first-out. This means that objects are spoken in the order in which they are received.

The *top of queue* item is the head of the queue. The top of queue item is the item currently being spoken or is the item that will be spoken next when a paused synthesizer is resumed.

The Synthesizer interface provides a number of methods for manipulating the output queue. The enumerateQueue method returns an Enumeration object containing a SynthesizerQueueItem for each object on the queue. The first object in the enumeration is the top of queue. If the queue is empty the enumerateQueue method returns null.

8.3.4 MONITORING SPEECH OUTPUT

All the speak and speakPlainText methods accept a SpeakableListener as the second input parameter. To request notification of events as the speech object is spoken an application provides a non-null listener.

Unlike a SynthesizerListener that receives synthesizer-level events, a SpeakableListener receives events associated with output of individual text objects: output of Speakable objects, output of URLs, output of JSML strings, or output of plain text strings.

The mechanism for attaching a SpeakableListener through the speak and speakPlainText methods is slightly different from the normal attachment and removal of listeners. There are, however, addSpeakableListener and removeSpeakableListener methods on the Synthesizer interface. These add and remove methods allow listeners to be provided to receive notifications of events associated with *all* objects being spoken by the Synthesizer.

The SpeakableEvent class defines eight events that indicate progress of spoken output of a text object. For each of these eight event types, there is a matching method in the SpeakableListener interface. For convenience, a SpeakableAdapter implementation of the SpeakableListener interface is provided with trivial (empty) implementations of all eight methods.

8.3.5 SYNTHESIZER PROPERTIES

8.3.5.1 SELECTING VOICES

The SynthesizerProperties interface extends the EngineProperties interface. The JavaBeans property mechanisms, the asynchronous application of property changing, and the property change event notifications are all inherited engine behavior and are described in that section.

The SynthesizerProperties object is obtained by calling the getEngineProperties method (inherited from the Engine interface) or the getSynthesizerProperties method. Both methods return the same object instance, but the latter is more convenient since it is an appropriately cast object.

The SynthesizerProperties interface defines five synthesizer properties that can be modified during operation of a synthesizer to effect speech output.

The *voice* property is used to control the speaking voice of the synthesizer. The set of voices supported by a synthesizer can be obtained by the getVoices method of the synthesizer's SynthesizerModeDesc object. Each voice is defined by a voice name, gender, age and speaking style. Selection of voices is described in more detail in *Selecting Voices*.

The remaining four properties control *prosody*. Prosody is a set of features of speech including the pitch and intonation, rhythm and timing, stress and other characteristics which affect the style of the speech. The prosodic features controlled through the SynthesizerProperties interface are:

Volume*:* a float value that is set on a scale from 0.0 (silence) to 1.0 (loudest).

***Speaking rate**:* a float value indicating the speech output rate in words per minute. Higher values indicate faster speech output. Reasonable speaking rates depend upon the synthesizer and the current voice (voices may have different natural speeds). Also, speaking rate is also dependent upon the language because of different conventions for what is a "word". For English, a typical speaking rate is around 200 words per minute.

Pitch: the baseline pitch is a float value given in Hertz. Different voices have different natural sounding ranges of pitch. Typical male voices are between 80 and 180 Hertz. Female pitches typically vary from 150 to 300 Hertz.

***Pitch range**:* a float value indicating a preferred range for variation in pitch above the baseline setting. A narrow pitch range provides monotonous output while wide range provides a more lively voice. The pitch range is typically between 20% and 80% of the baseline pitch.

8.3.5.2 PROPERTY CHANGES IN JSML

In addition to control of speech output through the SynthesizerProperties interface, all five-synthesizer properties can be controlled in JSML text provided to a synthesizer. The advantage of control through JSML text is that property changes can be finely controlled within a text document. By contrast, control of the synthesizer properties through the SynthesizerProperties interface is not appropriate for word-level changes but is instead useful for setting the default configuration of the synthesizer. Control of the

SynthesizerProperties interface is often presented to the user as a graphical configuration window.

Applications that generate JSML text should respect the default settings of the user. To do this, relative settings of parameters such as pitch and speaking rate should be used rather than absolute settings.

For example, users with vision impairments often set the speaking rate extremely high - up to 500 words per minute - so high that most people do not understand the synthesized speech. If a document uses an absolute speaking rate change (to say 200 words per minute which is fast for most users), then the user will be frustrated.

Changes made to the synthesizer properties through the SynthesizerProperties interface are persistent: they affect all succeeding speech output. Changes in JSML are explicitly localized (all property changes in JSML have both start and end tags).

8.4 CONCLUSION :

The Java Speech API is a freely available specification and therefore anyone is welcome to develop an implementation. The following implementations are known to exist:

Fleets

IBM's "Speech for Java"

The Cloud Garden

Lernout & Hauspie's TTS for Java Speech API

Conversa Web 3.0

Festival

Chapter 9

9. MAIL SERVER (JAVA MAIL API)

9.1 INTRODUCTION

JavaMail provides a common, uniform API for managing electronic mail. It allows service-providers to provide a standard interface to their standards-based or proprietary messaging systems using the Java programming language. Using thisAPI, applications access message stores, and compose and send messages.The JavaMail API is composed of a set of abstract classes that model the various pieces of a typical mail system. These classes include:

Message—Abstract class that represents an electronic mail message.
Java Mail implements the RFC822 and MIME Internet messaging standards. The MimeMessageclass extendsMessageto represent a MIME-style email message.
Store—Abstract class that represents a database of messages maintained by amail server and grouped by owner. A Store uses a particular access protocol
Folder—Abstract class that provides a way of hierarchically organizing messages.

Folders can contain messages and other folders. A mail server
Provides each user with a default folder, and users can typically create and fillSubfolders.

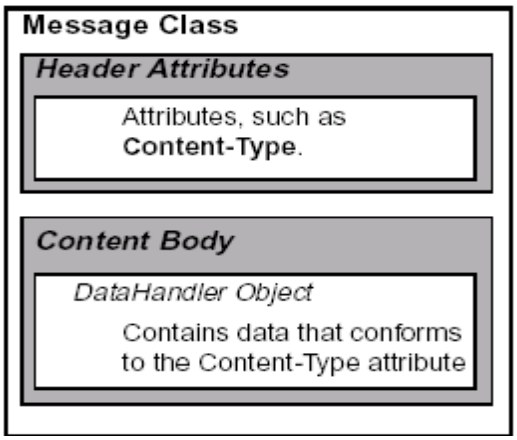
Transport—Abstract class that represents a specific transport protocol. A Transport object uses a particular transport protocol to send a message.

9.2 THE STRUCTURE OF A MESSAGE

The Messageclass models an electronic mail message. It is an abstract class that implements the Part interface.The Messageclass defines a set of attributes and content for an electronic mailmessage. The attributes, which are name-value pairs, specify addressing informationand define the structure of the message's content (its content type). Messages can contain a single content object or, indirectly, multiple content objects. In either case, thecontent is held by a DataHandlerobject.

9.2.1 SIMPLE MESSAGES

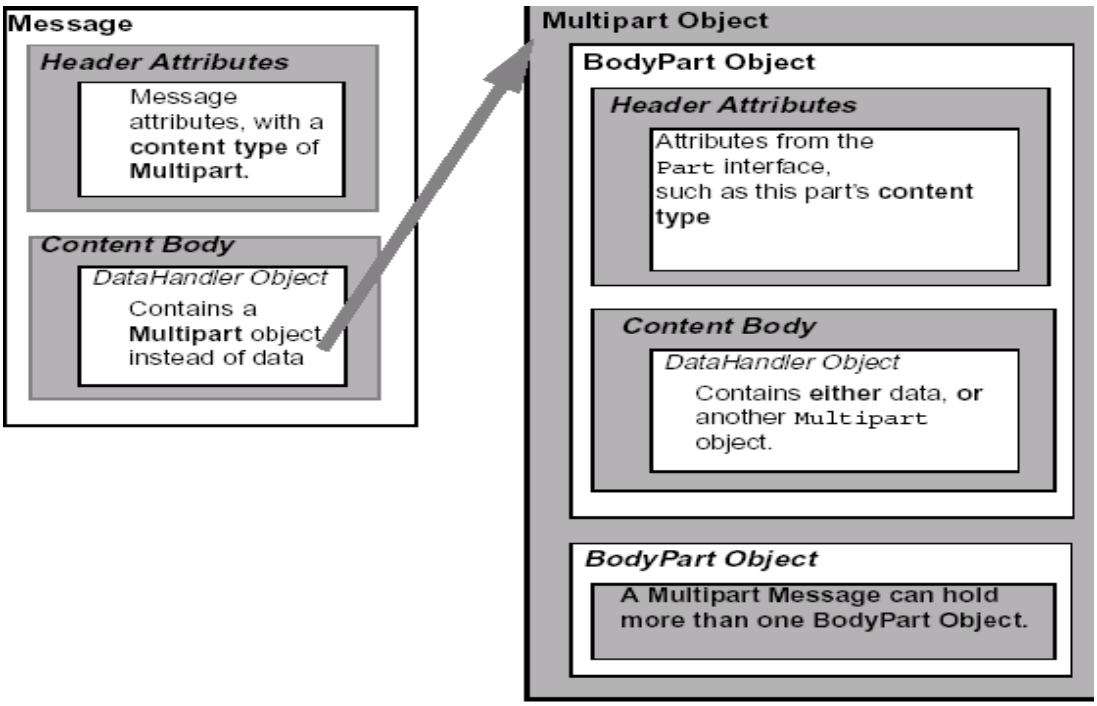
A simple message has a single content object, which is wrapped by a DataHandlerobject. The following figure shows the structure of a Message object:



9.2.2 MULTIPART MESSAGES

In addition to the simple structure shown above, messages can also contain multiple content objects. In this case theDataHandlerobject contains aMultipart object, instead of merely a a single block of content data.AMultipartobject is a container of BodyPartobjects. The structure of aBodyPartobject is similar to the structure of a

Message object, because they both implement the Partinterface.EachBodyPart object contains attributes and content, but the attributes of aBodypartobject are limited to those defined by the Part interface. An importantattribute is the content-type of this part of the message content. The content of aBodyPartobject is aDataHandler that contains either data or anotherMultipart object. The following figure shows this structure:



9.3 MESSAGES AND JAVABEANS ACTIVATION FRAMEWORK

A DataHandler object represents the content of a message. The DataHandler class is part of the JavaBeans Activation Framework (JAF).
The Data Handler class provides a consistent interface to data, independent of its source and format. The data can be from message stores, local files, URLs or objects in the Java programming language.

9.3.1 DATASOURCE

A DataHandler object accepts data in the form of an object in the Java programming language directly. For data from message stores, files or URLs, however, a DataHandler depends on objects that implement the Data Source interface to provide data access. A dataSource object provides access to data in the form of an input stream .The Data Sourceinterface is also part of the JAF. Java Mail provides the following Data Source objects:

javax.mail.MultipartDataSource

javax.mail.internet.MimePartDataSource

9.3.2 THE DATACONTENTHANDLER

DataHandler objects return the content of a message as an object in the Java programming language. They use objects that implement the DataContentHandlerinterface to translate message content between the streams provided by DataSource objects and objects in the Java programming language.

9.4 MESSAGE STORAGE AND RETRIEVAL

Users interact with message stores to fetch and manipulate electronic mail messages.This chapter discusses how to implement the classes that allow clients this access. Ifyou are creating a JavaMail service provider that allows a client to send mail, but doesnot interface with a mail store, you do not have to implement this functionality.

9.4.1 STORE

The Store class models a message database and its access protocol. A client uses it to connect to a particular message store, and to retrieve folders (groups of messages).To provide access to a message store, you must extend the Store class and implement its abstract methods. In addition, you must override the default implementation of at least one method that handles client authentication. The next sections cover how to write these methods. They begin with authentication, since it precedes retrieval when the provider is used.

9.4.1.1 AUTHENTICATION

JavaMail provides a framework to support both the most common style of authentication, (username, passphrase), and other more sophisticated styles such as a challenge-response dialogue with the user. To furnish the (username, passphrase) style authentication in your provider, override the protocolConnect method. Touse another style of authentication, you must override the version of the connectmethod that takes no arguments.

9.4.1.2 FOLDER RETRIEVAL

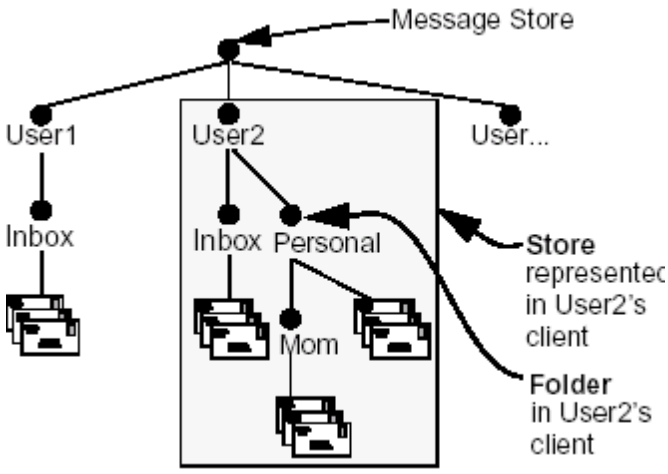
A message store stores messages, and often allows users to further group their messages. These groups of messages are called folders, and are represented by the abstract class, Folder. The Store class provides abstract methods for allowing the user to retrieve a folder:

- getDefaultFolder

- getFolder

9.4.2 FOLDERS

The Folder class models a node in a mail storage hierarchy. Folders can contain messages or subfolders or both. The following figure illustrates this:



JavaMail Guide for Service Providers August 1998

Each user has a folder that has the case-insensitive name INBOX. Providers must support this name. Folders have two states: they can be closed (operations on a closed folder are limited) or open.

Folders

Since Folder is an abstract class, you must extend it and implement its abstract methods. In addition, some of its methods have default implementations that,depending on your system, you may want to override for performance purposes. Thissection covers many of the abstract methods

that you must implement, and the methods whose default implementations you might want to override. It groups them in the following way:

- “Folder Naming”: getName, getFullName, getSeparator
- “Folder State”: open, close
- “Messages Within a Folder”: getMessage, getMessages, search, fetch
- “Folder Management”: getPermanentFlags, setFlags,appendMessages, copyMessages, expunge.

9.5 ADDRESS

The Address class is an abstract class. Subclasses provide specific implementations. Every Address subclass has a type-name, which identifies the address-type represented by that subclass. For example, the javax.mail.internet.InternetAddress subclass has the type-name: rfc822. The type-name is used to map address-types to Transport protocols. These mappings are set in the address.map registry. For example, the default address.map in the JavaMail package contains the following entry:
rfc822=smtp The Address-type to Transport mapping is used by JavaMail to determine the Transport object to be used to send a message. The getTransport(Address) method on Session does this, by searching the address.map for the transportprotocolthat corresponds to the type of the given address object. For example,invoking the getTransport(Address) method with an InternetAddress object,will return a Transport object that implements the smtp protocol.An Address subclass may also provide additional methods that are specific to that address-type. For example, one method that the InternetAddress class adds is the getAddress method.

9.6 EVENTS

The Store, Folder and Transport classes use events to communicate state changes to applications. The documentation for the methods of these classes specify whichevents to generate. A compliant provider must broadcast these events.To broadcast an event, call the appropriate notifyEventListeners method. Forexample, to manage MessageCountEvents for new mail notification, your Foldersubclass should call the notify MessageAddedListeners(msgs) method. (It is best to use the default implementations of the NotifyEventListeners methods,because they dispatch their events in an internal event-dispatcher thread. Using aseparate thread like this avoids deadlocks from breakage in the locking hierarchy.) Every event generated by the Store, Folder and Transport classes also has associated addListener and removeListener methods. Like the notifyEventListeners methods, these methods already have useful implementations. A programmer using your service provider implementation calls the appropriate addEventListener and removeEventListener methods to control which event notifications are received.

9.7 CONCLUSION

JavaMail clients must package provider software for use. To do this:

- Choose a suitable name for your package

The recommended way of doing this is to reverse your company domain name, and then add a suitable suffix. For example, Sun's IMAP provider is named com.sun.mail.imap.

- Make sure that your key classes are public

  If you provide access to a message store, your Store subclass must be a public class. If you provide a way to send messages, your Transport subclass must be a public class. (This allows JavaMail to instantiate your classes.)

- Bundle your provider classes into a suitably named jar file

  The name of the jar file should reflect the protocol you are providing. For example, an NNTP provider may have a jar file named nntp.jar. Refer to a suitable Java programming language book for details on how to create jar files.Because your jar file must be included in an application's classpath so that it canbe found by the application's classloader, include the name of your jar file in the documentation for your provider. Mention that the application's classpath should beupdated to include the location of the jar file.

- Create a registry entry for the protocol your implementation provides

  A registry entry is a set of attributes that describe your implementation. There are five attributes that describe a protocol implementation. Each attribute is a name-value pair whose syntax is name=value. The attributes are separated by semicolons (;).

CONCLUSION

The VMS server is the first server in Pakistan through which you can hear to your emails from anywhere around the world just by having a simple touch tone telephone or simple mobile. Our server is complete and in working phase. It can be used commercially which only requires an exchange card for multiple users, but there is no need for change in software for that. It can be used commercially.

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.

- **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.

- **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**

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