Java InstantCode: Developing Applications Using JCA

SkillSoft Press © 2004

This book describes how JCA defines a development methodology and suggests analysis and design patterns that are useful both for building connectors to legacy applications and for designing adapters for new applications.

Table of Contents

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

Java provides Java Communication Application Programming Interface (JCA), which contains classes and interfaces that help you develop platform-independent communication applications based on technologies, such as voice, mail, fax, or smart cards. JCA provides the javax.comm package, which contains interfaces, classes, and exception classes to interact with the RS232 serial and parallel ports.

This book describes how JCA defines a development methodology and suggests analysis and design patterns that are useful both for building connectors to legacy applications and for designing adapters for new applications.

About the Author

Vibha holds a Bachelor's degree in IT Engineering. She is proficient in technologies such as C++, Java, HTML, DHTML, J2EE, EJB, JNDI, JMS, and RMI. In addition, she has written books on Oracle, jDeveloper, and jBuilder for NIIT.

Credits

I would like to thank Reena Roy, S. Sripriya, Gaurav Bhatla, and Anurag for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

Copyright

Java InstantCode: Developing applications using JCA

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introducing the Java Communication API

Java provides Java Communication Application Programming Interface (JCA), which contains classes and interfaces that help you develop platform-independent communication applications based on technologies, such as voice, mail, fax, or smart cards. JCA provides the javax.comm package, which contains interfaces, classes, and exception classes to interact with the RS232 serial and parallel ports. The serial ports are also called COM ports.

Apart from serial and parallel ports, there exists other type of ports such as Universal Serial Bus (USB) ports. A USB port provides just one standard plug-in for all peripherals in comparison to serial or parallel ports, which provide more than one plug-in for individual devices. A USB port enables you to detect the USB enabled device without restarting your computer.

This chapter explains the RS232 standards for serial ports. It also explains JCA and its classes and interfaces, and describes how to install and set up JCA.

The RS232 Standard

RS232 is a standard that specifies how serial ports communicate. The javax.comm package enables you to develop applications that interact with the devices attached to RS232 serial ports. The RS232 communication standard is based on asynchronous serial communication, which means that communication between the sending device and the receiving device occurs bit by bit. The sending device needs to send the start bit and the stop bit to mark the start and the end of a message.

RS232 Bits Streams

According to the RS232 standard, the port driver breaks the data that needs to be sent into data words. The length of the data words can range from five to eight bits per word. To correctly synchronize the word transfer, the port driver adds some additional bits to the word for error checking. In addition, it is important that the number of bits sent by the sending device must be equal to number of bits received by the receiving device. The additional bits sent with the words are of various types, such as:

- Start bit: Represents the start of a data word. The start bit does not represent the data word that needs to be sent; it is the starting point of the data word. The space line recognizes the start bit.
- Data bit: Represents a bit from the data word. After sending the start bit, the communication application sends the data bits to a data word.
- Parity bit: Represents a bit in the data word for error detection. After sending the data word, the communication application automatically adds a parity bit to the data word. The sending device calculates the value of the parity bit based on the information that it sends. The receiving device also performs the same calculation as the sending device and checks if the actual parity bit value is the same as the calculated value.
- Stop bit: Represents the end of the data word. All data bits and parity bits are within the start and stop bits.

Physical Properties of the RS232 Cable

The RS232 standards define the physical properties of the RS232 cable, such as the voltage level at which the bit will be sent and the maximum length of the cable for communication to take place. The RS232 standards define two states of signal level, high bit and low bit. A negative voltage identifies the high bit state and a positive voltage identifies a low bit state. RS232 also defines standards for the maximum cable length. Per RS232 standards, the maximum cable length is 50 feet. This standard allows communication to occur at the optimum speed.

Error Detection

To detect errors during communication, both the sending and the receiving device calculate the value of the parity bit. If the value of the parity bit after calculation is the same at both ends, the communication is error-free; otherwise, it is not. It is mandatory for both the sender and the receiver to use the same algorithms to check the value of the parity bits sent with the data word. The two types of parity checking are:

- Even parity: In even parity, the number of data bits sent contains an even number of logical 1s. If the number of high data bits is even, the communication application adds a low parity to a communication message; otherwise, the communication application adds a high parity bit.
- Odd parity: The odd parity bit is similar to the even parity bit but in odd parity, the number of high bits is always odd.

Communicating Using the Serial Device

Serial devices communicate using a serial port, which transfers data one bit at a time using a single wire in a data cable. The serial port on a computer supports full-duplex communication. This means that the port can send and receive data at the same time. To send and receive data separately, serial ports use separate wires.

A device attached to a serial port starts communicating by first sending a start bit. The start bit signifies that the data or message is about to start. The device can send five, six, seven, or eight data bits, based on the agreed number of bits, between the device and the serial port. The device and a port must agree on the number of data bits to be sent and received. After data has been sent, the device sends a stop bit to indicate that the transfer of data bits is over. A stop bit has a value of one.

Communicating Using the Parallel Device

Parallel devices communicate using a parallel port, which simultaneously transfers eight bits over eight separate wires in a data cable. The parallel port consists of a connector with 17 signal lines and 8 ground lines. The signal lines include Control, Status, and Data lines.

The Control lines act as an interface control. These lines enable handshaking of signals from the computer to the device attached to a parallel port. The Status lines help in indicating the status of the device attached to the computer, such as printer busy, paper empty, and interface or peripheral errors. The Data lines provide data from the computer to the device attached to a parallel port, such as printer.

PREVIOUS NEXT

Functions of JCA

Using JCA, you can create objects of serial ports and parallel ports. Java applications use the objects to communicate with the ports. The various functions that you can perform using JCA are:

- Open ports for communication.
- Resolve port contention among Java objects.
- Perform asynchronous and synchronous communication with the Java application and the devices attached to the serial and parallel ports.
- Claim ownership or control of a port.
- List the ports available on a computer.
- Send data to and receive data from a port.

The javax.comm Package

JCA provides the javax.comm package, which is a Java extension Application Programming Interface (API). The standard Java Development Kit (JDK) does not provide this API. The javax.comm package provides classes and interfaces that can communicate with serial ports and parallel ports but cannot communicate with other ports, such as USB.

Classes

JCA defines three levels of classes: high-level, low-level, and driver-level classes. High-level classes manage the ownership of and access to communication ports. Low-level classes provide an interface to physical ports. Driver-level classes provide an interface between low-level classes and operation systems.

The javax.comm package contains various classes that you can use to develop applications to interact with the communication ports on a computer. The classes in the javax.comm package are:

- CommPort: Is an abstract class that describes the communication ports available on a particular computer. The CommPort class defines various methods that you can use to control input/output operations on ports. The most commonly used methods in the CommPort class are:
  - `close()`: Closes a communication port.
  - `disableRecieveFraming()`: Is an abstract method that disables receive framing.
  - `disableReceiveTimeout()`: Is an abstract method that disables receive timeout.
  - `enableRecieveFraming()`: Is an abstract method that enables receive framing. The enableRecieveFraming () method is driver-dependent. The method allows receive framing only if driver support enables receive framing.
  - `enableReceiveTimeout()`: Is an abstract method that enables receive time out. The enableReceiveTimeOut() method is driver-dependent.
  - `getInputBufferSize()`: Is an abstract method that returns the size of the input buffer. The getInputBufferSize() returns an integer value.
  - `getOutputBufferSize()`: Is an abstract method that returns the size of the output buffer. The getOutputBufferSize() returns an integer value.
  - `getInputStream()`: Is an abstract method that returns the input stream. The input stream is the only way to receive data from the communication port. The getInputStream() method returns an InputStream type value.
  - `getName()`: Retrieves a string that represents the name of the communication port.
  - `toString()`: Returns the string representation of a communication port.
  - `getOutputStream()`: Returns an output stream. This stream works as a mediator to send data to the communication port.
  - `isReceiveTimeoutEnabled()`: Is an abstract method that verifies whether or not receive timeout is enabled. The isReceiveTimeoutEnabled() method returns a Boolean value.
  - `isReceiveFramingEnabled()`: Is an abstract method that verifies whether or not receive framing is enabled. The isReceiveFramingEnabled() method returns a Boolean value.
  - `setOutputBufferSize()`: Is an abstract method that sets the size of the output buffer.
  - `setInputBufferSize()`: Is an abstract method that sets the size of the input buffer.
- CommPortIdentifier: Controls access and communication between a Java application and ports. The CommPortIdentifier class also defines methods for various operations, such as determining available communication ports, opening communication ports, and determining the ownership of ports. The commonly used methods in the CommPortIdentifier class are:
  - `addPortName()`: Adds the name of a port to the port name list.
  - `addPortOwnershipListener()`: Registers a Java application to receive event notification from the port when a change occurs in the ownership of the port. The three types of events are PORT_OWNED, PORT_UNOWNED, and PORT_OWNERSHIP_REQUESTED. The PORT_OWNED event specifies that a port has an owner. The PORT_UNOWNED event specifies that the port does not have an owner. The PORT_OWNERSHIP_REQUESTED event specifies that a Java application wants to relinquish ownership of the port.
  - `getCurrentOwner()`: Obtains the name of the current owner of a port.
  - `getName()`: Acquires the name of the port.
  - `getPortIdentifier()`: Obtains an enumeration object that contains a list of the available ports. The enumeration object contains objects of type CommPortIdentifier object.

- `getPortType()`: Returns an integer value that represents the port type.
- `open()`: Opens a port for communication, if the port is free. The open() method acquires exclusive ownership of the port if there is no other owner of the requested port.
- `isCurrentlyOwned()`: Checks the ownership of a port, whether it is true or false. If the requested port is owned by a device, the method returns true; and if the requested port is not owned, the method returns false.
- `removePortOwnershipListener()`: Unregisters the CommPortOwnershipListener listener.

- `ParallelPort`: Is a subclass of the CommPort class. The ParallelPort class describes a low-level interface to a parallel port. The most commonly used methods in the ParallelPort class are:
  - `addEventListener()`: Is an abstract method that registers an event listener for a parallel port. You can add only one event listener on one parallel port.
  - `removeEventListener()`: Is an abstract method that unregisters an event listener for a parallel port.
  - `getOutputBufferFree()`: Is an abstract method that retrieves the number of bytes available in the output buffer. The getOutputBufferFree() method returns an integer value.
  - `isPaperOut()`: Is an abstract method that verifies if the port indicates a printer is in the Out of Paper state. The isPaperOut() method returns a Boolean value. For example, the method returns true if the printer's tray does not contain paper; else, the method returns false.
  - `isPrinterBusy()`: Is an abstract method that verifies if the port indicates the Printer Busy state. The isPrinterBusy() method returns a Boolean value. For example, the method returns true if the printer is busy; else, the method returns false.
  - `isPrinterSelected()`: Is an abstract method that verifies if the port indicates a printer in the selected state. The isPrinterSelected() method returns a Boolean value. The method returns true if the printer is in the selected state; else, the method returns false.
  - `isPrinterTimedOut()`: Is an abstract method that verifies if the port indicates that a printer has timed out. The isPrinterTimedOut() method returns a Boolean value. The method returns true if the printer has timed out; else, the method returns false.
  - `notifyOnBuffer()`: Is an abstract method that conveys that the communication application should be notified when the output buffer is empty.
  - `notifyOnError()`: Is an abstract method that conveys that the communication application should be notified when an error occurs on a port.
  - `restart()`: Is an abstract method that resumes output after an error occurs.
  - `getMode()`: Is an abstract method that retrieves the currently configured mode of the port.
  - `setMode()`: Is an abstract method that sets the mode of a printer port. The different printer modes are LPT_MODE_SPP, LPT_MODE_PS2, LPT_MODE_EPP, and LPT_MODE_ECP.
  - `suspend()`: Is an abstract method that suspends output.
  - `removeEventListener()`: Is an abstract method that unregisters an event listener.
  - `isPrinterError()`: Is an abstract method that returns a Boolean value to specify if an error occurred on a printer.

- ParallelPortEvent: Defines the events of the parallel port. The ParallelPortEvent class defines some static variables, such as PAR_EV_ERROR and PAR_EV_BUFFER, which you can use within the methods defined in the class. The PAR_EV_ERROR variable indicates that an error has occurred at a port. The PAR_EV_BUFFER variable indicates that the port output buffer is empty. The methods defined in the ParallelPortEvent class are:
  - `getEventType()`: Returns an integer value that represents the type of event. The getEventType() method may return the PAR_EV_ERROR or PAR_EV_BUFFER static variable.
  - `getNewValue()`: Returns the new value of the changed state of a parallel port. The getNewValue() method returns a Boolean value.
  - `getOldValue()`: Returns the old Boolean value of the state change of a parallel port.

- SerialPort: Is a subclass of CommPort that provides access to a different serial port on a computer. In addition, the SerialPort class also defines methods that allow port to open and close and data to be sent and received. The SerialPort class also defines various static integer variables that the methods defined in the SerialPort class use. The most commonly used methods in the SerialPort class are:
  - `getBaudRate()`: Returns an integer value that represents the current baud rate of a serial port. The getBaudRate() method is an abstract method.

- `getDataBits()`: Returns an integer value that represents the current number of data bits configured on a serial port. The getDataBits() method is an abstract method. The return integer value can be equal to the static variables DATABITS_5, DATABITS_6, DATABITS_7, or DATABITS_8. Table 1-1 describes these variables.
- `getFlowControlMode()`: Returns an integer value that represents the current flow control mode configured on a serial port. The getFlowControlMode() method is an abstract method.
- `getParity()`: Returns an integer value that represents the current parity setting that is configured. The getParity() method is an abstract method.
- `getStopBits()`: Returns an integer value that represents the currently defined stop bits. The getStopBits() method is an abstract method. The returned integer can be equal to the value of the STOPBITS_1, STOPBITS_2, or STOPBITS_3 static variables. Table 1-1 describes the static variables.
- `isCD()`: Is an abstract method that retrieves the state of the Carrier Detect (CR) bit in Universal Asynchronous Receiver/Transmitter (UART). The isCD() method returns a Boolean value.
- `isCTS()`: Is an abstract method that retrieves the state of the Clear To Send (CTS) bit in UART. The isCTS() method returns a Boolean value.
- `isDSR()`: Is an abstract method that retrieves the state of the Data Set Ready (DSR) bit in UART. The isDSR() method returns a Boolean value.
- `isDTR()`: Is an abstract method that retrieves the state of the Data Terminal Ready (DTR) bit in UART. The isDTR() method returns a Boolean value.
- `isRI()`: Is an abstract method that retrieves the state of the Ring Indicator (RI) bit in UART. The isRI() method returns a Boolean value.
- `isRTS()`: Is an abstract method that retrieves the state of the Request To Send (RTS) bit in UART. The isRTS() method returns a Boolean value.
- `notifyOnBreakInterrupt()`: Is an abstract method that conveys that the communication application must be receive notification from the port when there is a break interrupt on the line.
- `notifyOnCarrierDetect()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Carrier Detect (CD) bit changes.
- `notifyOnCTS()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Clear To Send (CTS) bit changes.
- `notifyOnDataAvailable()`: Is an abstract method that conveys that the communication application must receive event notification from the port when input data is available.
- `notifyOnDSR()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Data Set Ready (DSR) bit changes.
- `notifyOnFramingError()`: Is an abstract method that conveys that the communication application must receive event notification from the port when a framing error occurs.
- `notifyOnOutputEmpty()`: Is an abstract method that conveys that the communication application must receive event empty.
- `notifyOnOverrunError()`: Is an abstract method that conveys that the communication application must receive event notification from the port when an overrun error occurs.
- `notifyOnParityError()`: Is an abstract method that conveys that it must receive event notification from the port when a parity bit error occurs.
- `notifyOnRingIndicator()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the RI bit changes.
- `removeEventListener()`: Is an abstract method that unregisters the registered event.
- `setDTR()`: Is an abstract method that sets the Data Terminal ready (DTR) bit in URAT.
- `setFlowControlMode()`: Is an abstract method that retrieves the current flow control mode.
- `setRTS()`: Is an abstract method that sets the DTR bit in URAT.

The SerialPort class also defines some static variables that the methods defined in the SerialPort class use, as described in Table 1-1:

Table 1-1: Static Variables of the SerialPort Class

Variable Name	Description
DATABITS_5	Represents 5-data bit format.
DATABITS_6	Represents 6-data bit format.
DATABITS_7	Represents 7-data bit format.

DATABITS_8	Represents 8-data bit format.
STOPBITS_1	Represents that the number of STOP bits is 1.
STOPBITS_2	Represents that the number of STOP bits is 2.
STOPBITS_1_5	Represents that the number of STOP bits is 1-1/2.
PARITY_NONE	Represents that there is no parity bit.
PARITY_ODD	Represents an ODD parity scheme.
PARITY_EVEN	Represents an EVEN parity scheme.
PARITY_MARK	Represents a MARK parity scheme.
PARITY_SPACE	Represents a SPACE parity scheme.
FLOWCONTROL_NONE	Represents that flow control is off.
FLOWCONTROL_RTSCTS_IN	Represents RTC/CTS flow control on input.
FLOWCONTROL_RTSCTS_OUT	Represents RTC/CTS flow control on output.
FLOWCONTROL_XONXOFF_IN	Represents XON/XOFF flow control on input.
FLOWCONTROL_XONXOFF_OUT	Represents XON/XOFF flow control on output.

- SerialPortEvent: Defines the events of the serial port. The various methods defined in the SerialPortEvent class are:
    - `getEventType()`: Returns an integer in the form of a static variable that represents the type of event, such as Break Interrupt (BR), Framing Error (FE), Overrun Error (OR), OUTPUT_BUFFER_EMPTY, DATA_AVAILABLE, or Parity Error (PE). For example, the DATA_AVAILABLE variable indicates that data is available on the serial port.
    - `getNewValue()`: Returns a new value for the state change of a serial port. The getNewValue() method returns a Boolean value.
    - `getOldValue()`: Returns the old value of the state change of a serial port. The getOldValue() method returns a Boolean value.

Interfaces

The javax.comm package provides interfaces that you can use to develop applications. You use the interfaces to communicate with the communication ports. The various interfaces defined in the javax.comm package are:

- CommDriver
- CommPortOwnershipListener
- ParallelPortEventListener
- SerialPortEventListener

The CommDriver interface is a part of a loadable device driver interface. The methods defined in the CommDriver interface are:

- `initialize()`: Registers the port name with the CommPortIdentifier class. In addition, the initialize() method also loads any required native libraries. The initialize() method is an abstract method.
- `getCommPort()`: Returns an object that extends the SerialPort or ParallelPort class. The open() method of the CommPortIdentifier class invokes the getCommPort() method.

The CommPortOwnershipListener interface broadcasts various port ownership events across an application. Opening a port triggers and broadcasts the CommPortOwnershipListener event of type PORT_OWNED. Closing a port triggers and broadcasts the CommPortOwnershipListener event of type PORT_UNOWNED.

The ParallelPortEventListener interface broadcasts the events of parallel ports using the parallelEvent() method. The parallelEvent() is the only method defined in the ParallelPortEventListener interface.

The SerialPortEventListener interface broadcasts the events of serial ports using the serialEvent() method. The serialEvent() is the only method defined in the SerialPortEventListener interface.

Exception Handling

The javax.comm package provides three exception classes that you can use to develop communication applications. An exception class catches the different exceptions that a communication application throws when you use the javax.comm package. Table 1-2 describes various exception classes:

Table 1-2: Exception Classes

Exceptions Classes	Description
NoSuchPortException	Catches the exception that the Java application throws when the application is unable to find the specified port.

PortInUseException	Catches the exception that the Java application throws if the specified port is in use.
UnsupportedCommOperationException	Catches the exception that the Java application throws when the port driver does not allow you to perform a specified operation on the port.

PREVIOUS NEXT

How to Install JCA

To use the features that JCA provides, you need to configure the javax.comm package and the setup environment to execute your communication applications. To install JCA and configure the package setting:

1. Download the Javacomm20-win32 JCA, which is available in zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip javacomm20-win32.zip in the C drive.

3. Copy win32comm.dll to the %JAVA_HOME%\bin directory. %JAVA_HOME% represents the directory that stores JDK on the computer. To copy win32comm.dll, specify the following command at the command prompt:

   ```
   C:\>copy c:\commapi\win32com.dll %JAVA_HOME%\bin
   ```

4. Copy the comm.jar file to the %JAVA_HOME%\lib directory by specifying the following command at the command prompt:

   ```
   C:\>copy c:\commapi\comm.jar %JAVA_HOME%\lib
   ```

5. Copy javax.comm.properties to the %JAVA_HOME%\lib directory by specifying the following command at the command prompt:

   ```
   C:\>copy c:\commapi\javax.comm.properties %JAVA_HOME%\lib
   ```

6. Add the comm.jar file to your classpath by specifying the following command at the command prompt:

   ```
   set classpath = %classpath%;%JAVA_HOME\lib; %JAVA_HOME\jre\lib\ext\comm.jar;
   ```

Chapter 2: Creating a Port Information Application

The Java Communication API (JCA) supports the javax.comm package, which provides the CommPortIdentifier class to control access to communications ports. For example, you can determine the ports attached to a computer, open any specific port for Input/Output (I/O) operations, view the description of the port, and determine the ownership of the port by using methods in the CommPortIdentifier class.

This chapter explains how to develop a Port Information application, which uses the javax.comm package to list all the serial and parallel ports attached to a computer and provide information on ports selected by an end user. It also explains how to open the ports for Input/Output (I/O) operations.

Architecture of the Port Information Application

The Port Information application uses the following files:

- PortDescription.java: Creates a user interface that an end user can use to list the parallel and serial ports, open a selected port, and view the description of the selected port.
- ShowDescriptionWindow.java: Displays the tabular description of the selected serial or parallel port.

Figure 2-1 shows the architecture of the Port Information application:

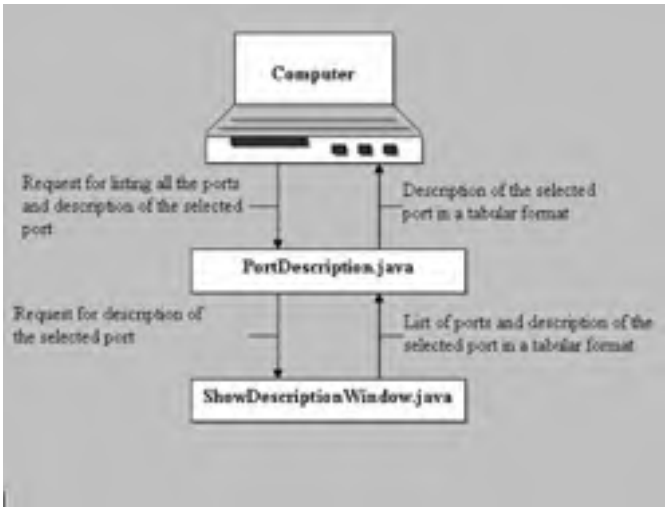

Figure 2-1: Architecture of the Port Information Application

The PortDescription.java file calls the ShowDescriptionWindow.java file, when the end user selects a serial or parallel port from the populated list of ports and clicks the Port Description button.

The ShowDescriptionWindow.java file displays the description of the selected port. The ShowDescriptionWindow.java file displays the various fields related to the selected port which includes Port Type, Port Name, Port Mode, Port Status, and Owner Name to the end user.

Creating the User Interface

The PortDescription.java file creates a user interface, which lets the end user list all serial and parallel ports attached to a computer and open any of the selected port for input/output operations using the CommPortIdentifier class. The PortDescription.java file uses the CommPort and ParallelPort classes to enable the end user to communicate through the selected port opened by the CommPortIdentifier class.

Listing 2-1 shows the code of the PortDescription.java file:

Listing 2-1: The PortDescription.java File

```
/* Imports required Comm classes. */
import javax.comm.*;
/* Imports required I/O classes. */
import java.io.*;
/* Imports required AWT classes. */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
/* Imports required Util classes. */
import java.util.*;
/* Class PortDescription - This class is the main class of the application.
This class initializes the interface and loads all components like the button and
listbox before displaying the result.
Constructor:
PortDescription - This constructor creates GUI.
Methods:
getPortDescription - This method gives port description.
openPort - This method opens a selected port.
getSerialPorts - This method fills list of serial ports.
getParallelPorts - This method fills list of parallel ports.
emptyList - This method clears a list.
main - This method creates the main window of the application and displays all the components.
*/
public class PortDescription implements ActionListener,ListSelectionListener
{
   /* Declare object of JFrame class. */
   JFrame frame;
   /* Declare object of Enumeration class. */
   Enumeration pList;
   /* Declare objects of JLabel class. */
   private JLabel apppagetitle;
   private JLabel seriallabel;
   private JLabel parallallabel;
   /* Declare objects of JButton class. */
   private JButton serialportbutton;
   private JButton commportbutton;
   private JButton openportbutton;
   private JButton portdescriptionbutton;
   /* Declare objects of JList class. */
   private JList listserial;
   private JList listcomm;
   /*
   Declare objects of DefaultListModel class.
   */
   private DefaultListModel listModelserial;
   private DefaultListModel listModelcomm;
   int descriptionpos;
   int buttonwidth=140;
   int buttonheight=25;
   public PortDescription()
   {
      /*
      Initialize and set the look and feel of the application to Windows Look and Feel.
      */
       try
      {
          UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         /*
         If an error occurs while loading the Windows Look and Feel, an
         error is printed and the application is closed.
         */
         System.out.println("Error setting window environment: " + e);
      }
      try
      {
         /*
         Initialize the object of the JFrame class and set the Title.
         */
         frame=new JFrame("Ports Application");
```

```
/*
Initialize the object of the JButton class and set the Title.
*/
Container pane=frame.getContentPane();
/*
Set the layout of the frame as Null.
*/
pane.setLayout(null);
/*
Set background color as white.
*/
pane.setBackground(Color.white);
/*
Initialize a new button, Set hot key as 'S' and add to content pane.
*/
serialportbutton=new JButton("List Serial Ports");
serialportbutton.setMnemonic('S');
serialportbutton.setActionCommand("serial");
serialportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
serialportbutton.addActionListener(this);
pane.add(serialportbutton);
/*
Initialize a new button, Set hot key as 'C' and add to content pane.
*/
commportbutton=new JButton("List Parallel Ports");
commportbutton.setMnemonic('C');
commportbutton.setActionCommand("parallel");
commportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
commportbutton.addActionListener(this);
pane.add(commportbutton);
/*
Initialize a new button, Set hot key as 'O' and add to content pane.
*/
openportbutton=new JButton("Open Port");
openportbutton.setMnemonic('O')
openportbutton.setActionCommand("open");
openportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
openportbutton.addActionListener(this);
pane.add(openportbutton);
/*
Initialize a new button, Set hot key as 'D' and add to content pane.
*/
portdescriptionbutton=new JButton("Port Description");
portdescriptionbutton.setMnemonic('D');
portdescriptionbutton.setActionCommand("description");
portdescriptionbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
portdescriptionbutton.addActionListener(this);
pane.add(portdescriptionbutton);
openportbutton.setEnabled(false);
/* Initialize a new label and add to content pane. */
seriallabel=new JLabel("Serial Ports");
seriallabel.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(seriallabel);
/* Initialize a new label and add to content pane. */
parallallabel=new JLabel("Parallel Ports");
parallallabel.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(parallallabel);
/*
Initialize a new label and add to content pane.
*/
apppagetitle=new JLabel("Port Information Application");
apppagetitle.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(apppagetitle);
/*
Declare and initialize the object of Insets class.
*/
Insets insets = pane.getInsets();
/*
Initialize the object of DefaultListModel class.
*/
listModelserial = new DefaultListModel();
/*
Initialize the object of DefaultListModel class.
*/
listModelcomm = new DefaultListModel();
/*
Initialize the object of JList class.
*/
listserial = new JList(listModelserial);
listcomm=new JList(listModelcomm);
/*
Set the selection mode of listserial as single selection.
Declare and initialize the object of JScrollPane. Set dimension.
Add list to scroll pane and scroll pane to frame's content pane.
*/
listserial.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
listserial.addListSelectionListener(this);
listserial.setVisibleRowCount(10);
```

```
   JScrollPane listScrollPane = new JScrollPane(listserial);
   listScrollPane.setPreferredSize(new Dimension(250,200));
   Dimension size=listScrollPane .getPreferredSize();
   listScrollPane.setBounds(80, 90, size.width, size.height);
   pane.add(listScrollPane);
   /*
   Set the selection mode of listcomm as single selection.
   Declare and initialize the object of JScrollPane.
   Set dimension. Add list to scroll pane add scroll pane to frame's content pane.
   */
   listcomm.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   listcomm.addListSelectionListener(this);
   listcomm.setVisibleRowCount(10);
   listScrollPane = new JScrollPane(listcomm);
   listScrollPane.setPreferredSize(new Dimension(250,200));
   size=listScrollPane .getPreferredSize();
   listScrollPane.setBounds(380, 90, size.width, size.height);
   pane.add(listScrollPane);
   /*
   Declare object of Dimension class and initialize it with getScreenSize() method.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   /*
   Set location at which window will be displayed.
   */
   frame.setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   /* Initialize left margin. */
   descriptionpos=70;
   size = serialportbutton.getPreferredSize();
   /*
   Set position of the serialportbutton button.
   */
   serialportbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height);
   descriptionpos=descriptionpos+size.width;
   size = commportbutton.getPreferredSize();
   /*
   Set position of the commportbutton button.
   */
   commportbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height);
   descriptionpos=descriptionpos+size.width;
   size = portdescriptionbutton.getPreferredSize();
   /*
   Set position of the portdescriptionbutton button.
   */
   portdescriptionbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height
   descriptionpos=descriptionpos+size.width;
   size = openportbutton.getPreferredSize();
   /*
   Set position of the openportbutton button.
   */
   openportbutton.setBounds(descriptionpos+ 5, 500 + insets.top, size.width, size.height);
   size = seriallabel.getPreferredSize();
   /*
   Set position of the seriallabel label.
   */
   seriallabel.setBounds(80 + insets.left, 60 + insets.top, size.width, size.height);
   size = parallallabel.getPreferredSize();
   /*
   Set position of the parallallabel label.
   */
   parallallabel.setBounds(380 + insets.left, 60 + insets.top, size.width, size.height);
   size=apppagetitle.getPreferredSize();
   /*
   Set position of the apppagetitle label.
   */
   apppagetitle.setBounds(250 + insets.left,  insets.top, size.width, size.height);
   /*
   Set the size of the Application frame.
   */
   frame.setSize(700,600);
   frame.setVisible(true);
   /*
   addWindowListener - It contains the windowClosing() method.
   windowClosing: It is called when the user clicks the cancel button of the Window. It closes
   Parameter:
   we - Object of WindowEvent class.
   Return Value: NA
   */
   frame.addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent we)
      {
         System.exit(0);
      }
   });
}
catch(Exception e)
{
```

```
            System.err.println(e);
            e.printStackTrace();
        }
    }
    /*
    valueChanged: It is called when the user selects a listitem from the list.
    Parameter:
    le - Object of ListSelectionEvent class.
    Return Value: NA
    */
    public void valueChanged(ListSelectionEvent le)
    {
        if(le.getSource()==listcomm)
        {
            if (listserial.getSelectedIndex()!=-1)
            {
                if (le.getValueIsAdjusting())
                {
                    listserial.clearSelection();
                }
            }
        }
        else if(le.getSource()==listserial)
        {
            if(listcomm.getSelectedIndex()!=-1)
            {
                if (le.getValueIsAdjusting())
                {
                    listcomm.clearSelection();
                }
            }
        }
    }
    /*
    actionPerformed - This method is called when the user clicks the Get Serial Ports,
    Get Comm Ports, Open Port or Port Description button.
    Parameters:
    ae - An ActionEvent object containing details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent e)
    {
        /*
        This is executed when the end user clicks the Get Serial Ports button.
        */
        if ("serial".equals(e.getActionCommand()))
        {
            getSerialPorts();
        }
        /*
        This is executed when the end user clicks the Get Parallel Ports button.
        */
        else if("parallel".equals(e.getActionCommand()))
        {
            getParallelPorts();
        }
        /*
        This is executed when the end user clicks the Port Description button.
        */
        else if("description".equals(e.getActionCommand()))
        {
            getPortDescription();
        }
        /*
        This is executed when the end user clicks the Open Port button.
        */
        else if("open".equals(e.getActionCommand()))
        {
            openPort();
        }
    }
    /*
    getPortDescription - This method open a new window to show port description of selected port.
    Parameters: NA
    Return Value: NA
    */
    public void getPortDescription()
    {
        /*
        Declare and initialize String objects.
        */
        String name="port";
        String isportopen="Not Open";
        /*
        Declare and initialize CommPort objects.
        */
        CommPort port =null;
        ShowDescriptionWindow descriptionwindow;
```

```
   descriptionwindow=new ShowDescriptionWindow();
   /*
   If the end user selected a port from serial port list
   */
   if (listserial.getSelectedIndex()!=-1)
   {
      name= listModelserial.getElementAt(listserial.getSelectedIndex()).toString();
      descriptionwindow.setPortType("Serial port ");
   }
   /*
   Else, the user selected a port from Parallel port list
   */
   else if(listcomm.getSelectedIndex()!=-1)
   {
      name = listModelcomm.getElementAt(listcomm.getSelectedIndex()).toString();
      descriptionwindow.setPortType("Parallel port ");
   }
   try
   {
      CommPortIdentifier commport = javax.comm.CommPortIdentifier.getPortIdentifier(name);
      try
      {
         /* Try to open port. */
         port= commport.open("PortDescription",20);
         descriptionwindow.isPortOpen("Not Open");
         if(commport.getPortType() == CommPortIdentifier.PORT_PARALLEL)
         {
            ParallelPort pport = (ParallelPort)port;
            int mode = pport.getMode();
            switch (mode)
            {
               case ParallelPort.LPT_MODE_ECP:
               descriptionwindow.setPortMode("ECP");
               break;
               case ParallelPort.LPT_MODE_EPP:
               descriptionwindow.setPortMode("EPP");
               break;
               case ParallelPort.LPT_MODE_NIBBLE:
               descriptionwindow.setPortMode("Nibble Mode");
               break;
               case ParallelPort.LPT_MODE_PS2:
               descriptionwindow.setPortMode("Byte mode");
               break;
               case ParallelPort.LPT_MODE_SPP:
               descriptionwindow.setPortMode("Compatibility mode");
               break;
               default:
            }
            if (port!=null)
            {
               port.close();
            }
         }
      }
      catch(PortInUseException pe)
      {
         descriptionwindow.isPortOpen("Open");
         /*
         Declare the String object and initialize it with a value returned by the getCurrentOwner
         */
         String ownername = commport.getCurrentOwner();
         if( ownername == null)
         {
            descriptionwindow.ownerName("unidenified");
         }
         else
         {
            descriptionwindow.ownerName(ownername);
         }
      }
      descriptionwindow.portName(commport.getName());
      descriptionwindow.showMessage(frame);
      descriptionwindow.setResizable(false);
   }
   catch(NoSuchPortException e)
   {
      descriptionwindow.portNotExists("yes");
   }
}
/*
openPort - This method try to open selected port for communication, if port is in
use or does not exists then this displays an error message to the end user.
Parameters: NA
Return Value: NA
*/
public void openPort()
{
   /* Declare and initialize String objects. */
```

```
      String name="";
      String porttype="Unknown port ";
      CommPort port =null;
      if (listserial.getSelectedIndex()!=-1)
      {
         name= listModelserial.getElementAt(listserial.getSelectedIndex()).toString();
         porttype="Serial port ";
      }
      else if(listcomm.getSelectedIndex()!=-1)
      {
         name = listModelcomm.getElementAt(listcomm.getSelectedIndex()).toString();
         porttype="Parallel port ";
      }
      try
      {
         CommPortIdentifier commport = javax.comm.CommPortIdentifier.getPortIdentifier(name);
         try
         {
            port= commport.open("PortListOpen",20);
            int userchoice=JOptionPane.showConfirmDialog(null,porttype+name+" has been
            successful opened. Click Yes to close this port.","Open Port",JOptionPane.YES_NO_OPTION)
            if (userchoice==0)
            {
               if (port!=null)
               {
                  port.close();
               }
            }
         }
         catch(PortInUseException pe)
         {
            String ownername = commport.getCurrentOwner();
            if( ownername == null)
            {
               JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+".
               This is owned by unidentified application","Open Port",JOptionPane.PLAIN_MESSAGE);
            }
            else
            {
               if ("Port currently not owned".equals(ownername))
               {
                  JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+". "
                  +ownername+".","Open Port",JOptionPane.PLAIN_MESSAGE);
               }
               else
               {
                  JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+". Port owned
                  +ownername,"Open Port",JOptionPane.PLAIN_MESSAGE);
               }
            }
         }
      }
      catch(NoSuchPortException e)
      {
         JOptionPane.showConfirmDialog(null,e,"Open Port",JOptionPane.YES_NO_OPTION);
      }
   }
   /*
   getSerialPorts - This method uses CommPortIdentifier API to get serial port names.
   Parameters: NA
   Return Value: NA
   */
   public void getSerialPorts()
   {
      emptyList("serial");
      pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
         CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
         if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
            /* Fill list from Serial ports. */
            listModelserial.addElement(cpi.getName());
         }
      }
      enableDisableOpenButton();
   }
   /*
   getParallelPorts - This method uses CommPortIdentifier API to get parallel port names.
   Parameters: NA
   Return Value: NA
   */
   public void getParallelPorts()
   {
      emptyList("parallel");
      pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
```

```
            CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
            if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
            {
               /* Fill list from Serial ports. */
               listModelcomm.addElement(cpi.getName());
            }
         }
         enableDisableOpenButton();
      }
      /*
      emptyList - This method clears all list items from the list box.
      Parameters: listtype - object of the String type.
      Return Value: NA
      */
      public void emptyList(String listtype)
      {
         if (listtype=="serial")
         {
            listModelserial.clear();
         }
         else if(listtype=="parallel")
         {
            listModelcomm.clear();
         }
      }
      /*
      enableDisableOpenButton - This method disables/enables open port button
      if serial and/or parallel port lists are empty/non-empty.
      Parameters: NA
      Return Value: NA
      */
      public void enableDisableOpenButton()
      {
         if ((listModelcomm.getSize()>0) || (listModelserial.getSize()>0))
         {
            openportbutton.setEnabled(true);
         }
         else
         {
            openportbutton.setEnabled(false);
         }
      }
      /*
      Main method that creates the instance of the PortDescription class.
      */
      public static void main(String[] args)
      {
         PortDescription pd=new PortDescription();
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortDescription class. This class generates the main window of the Port Information application, as shown in Figure 2-2:

Figure 2-2: The Port Information Application User Interface

When an end user clicks any button on the Ports Application window, the Port Information application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate method, based on the button that the end user clicks.

For example, when the end user clicks the List Serial Ports button, the actionPerformed() method invokes the getSerialPorts() method. The getSerialPorts() method calls the getPortIdentifiers() method of the CommPortIdentifier class to get the enumeration of ports attached to a computer. Next, the getSerialPorts() method calls the getPortType() method of the CommPortIdentifier class to check for the serial port from the enumeration of ports. If the getPortType() method is equal to the CommPortIdentifier.PORT_SERIAL variable, the port is assumed to be a serial port. The method then identifies all the serial ports attached to a computer and lists the ports in the Serial Ports list box.

The List Parallel Ports button functions in a similar manner as that of the List Serial Ports button.

When an end user clicks the Open Port button, the actionPerformed() method invokes the openPort() method. The openPort() method calls the open() method of the CommPortIdentifier class to open the selected port for various input/output operations. If the selected port is already in use, the Port Information application throws an exception.

When the end user clicks the Port Description button, the actionPerformed() method invokes the getPortDescription() method. The getPortDescription() method retrieves the information of the selected port using the CommPortIdentifier class and creates an object of the ShowDescriptionWindow.java class to display the following information about the selected port:

- Port Type
- Port Name
- Port Mode
- Port Status
- Owner Name

◄ PREVIOUS NEXT ►

Displaying the Port Description

The ShowDescriptionWindow.java file displays the name, type, mode, status, and owner of the selected port get from the PortDescription.java file. The ShowDescriptionWindow.java file displays the description of the selected port in a tabular format.

Listing 2-2 shows the contents of the ShowDescriptionWindow.java file:

Listing 2-2: The ShowDescriptionWindow.java File

```
/* Imports required AWT classes. */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
/*
class ShowDescriptionWindow -
This class is the sub class of the application.
This class displays the table of port description.
*/
class ShowDescriptionWindow extends JFrame
{
   /* Declare the String objects. */
   private String porttype;
   private String portstatus;
   private String ownername;
   private String portname;
   private String portexists;
   private String pmode;
   Border tableborder;
   /* Declare the JButton object. */
   JButton okbutton;
   /* Declare the JTable object. */
   JTable table;
   /*
   Declare the default constructor of the FileList class.
   */
   public ShowDescriptionWindow()
   {
      /* Set title of parent frame. */
      super("Port Description");
      porttype="Unknown port";
      portstatus="Not Open";
      ownername="";
      portname="";
      portexists="No";
      pmode="";
   }
   /*
   setPortType - This method sets the value of porttype field.
   Parameters: ptype - a String object containing port type.
   Return Value: NA
   */
   public void setPortType(String ptype)
   {
      porttype=ptype;
   }
   /*
   isPortOpen - This method sets the value of portstatus field.
   Parameters: openornot - a String object containing YES or NO.
   Return Value: NA
   */
   public void isPortOpen(String openornot)
   {
      portstatus=openornot;
   }
   /*
   ownerName - This method sets the value of ownername field.
   Parameters: openornot - a String object containing owner name.
   Return Value: NA
   */
   public void ownerName(String oname)
   {
      ownername=oname;
   }
   /*
   portName - This method sets the value of portname field.
   Parameters: pname - a String object containing port name.
   Return Value: NA
   */
   public void portName(String pname)
   {
      portname=pname;
```

```
}
public void setPortMode(String portmode)
{
   pmode=portmode;
}
/*
portNotExists- This method sets the value of portexists field.
Parameters: exists - A String object containing information of port validity.
Return Value: NA
*/
public void portNotExists(String exists)
{
   portexists=exists;
}
/*
enableDisableOpenButton - This method opens a window
which contains a port description table and ok button.
Parameters:
pd - a JFrame object.
Return Value: NA
*/
public void showMessage(JFrame pd)
{
   /*
   Initialize and set the look and feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error setting window LAF: " + e);
   }
   /* Get the content pane of this class. */
   Container messagewindowpane=getContentPane();
   /*
   Set the layout of the frame as Null.
   */
   messagewindowpane.setLayout(null);
   /*
   Set the background color as white.
   */
   messagewindowpane.setBackground(Color.white);
   /*
   Initialize a new button and add to content pane.
   */
   okbutton=new JButton("OK");
   messagewindowpane.add(okbutton);
   okbutton.setActionCommand("ok");
   okbutton.addActionListener(new ActionListener()
   {
      /*
      actionPerformed - This method is called when the user clicks the Ok button.
      Parameters:
      ae - an ActionEvent object containing details of the event.
      Return Value: NA
      */
      public void actionPerformed(ActionEvent e)
      {
          ShowDescriptionWindow.this.dispose();
      }
   });
   addWindowListener(new WindowAdapter()
   {
      /*
      windowClosing - This method is called when the user clicks the X button on the top bar.
      Parameters:
      e - an WindowEvent object containing details of the event.
      Return Value: NA
      */
      public void windowClosing(WindowEvent e)
      {
          ShowDescriptionWindow.this.dispose();
      }
   });
   /*
   Declare a Point object and initialize it with the getLocation() method
   */
   Point p=pd.getLocation();
   /*
   Set default location for this frame.
   */
   setLocation((int)p.getX(), (int)p.getY());
   /*
   Declare Dimension object and initialize it with the getPreferredSize() method.
   */
   Dimension size=okbutton.getPreferredSize();
```

```
        okbutton.setBounds(140, 180, size.width, size.height);
        /*
        Declare and initialize the Object array.
        */
        Object[][] rowdata={{" Port Type",porttype},{" Port Name",portname},
        {" Port Mode",pmode},{" Port Status",portstatus},{" Owner Name",ownername}};
        /*
        Declare and initialize the String array.
        */
        String[] columnname={"Text","Description"};
        table=new JTable(rowdata,columnname);
        messagewindowpane.add(table);
        size=table.getPreferredSize();
        tableborder=new EtchedBorder(EtchedBorder.RAISED);
        table.setBorder(tableborder);
        table.setBounds(50, 28, 250, size.height);
        setSize(350,250);
        setVisible(true);
    }
}
```

Download this listing.

The above listing displays the description of the selected port. The ShowDescriptionWindow.java file creates a table that displays the description of the selected port.

◄ PREVIOUS NEXT ►

Unit Testing

To test the Port Information application:

1. Download the JCA, which is available in a zip format with the name Javacomm20-win32.zip. The JCA can be downloaded from the following URL: http://java.sun.com/products/javacomm/downloads/index.html
2. Unzip the zip file downloaded from the above URL.
3. Copy the win32com.dll from the unzipped file to jre\bin directory where the J2SDK is installed.
4. Copy the comm.jar from the unzipped file to the jre\lib\ext directory where the J2SDK is installed.
5. Copy the javax.comm.properties from the unzipped file to the jre\lib directory where the J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:
   ```
   set path = %path%;D:\j2sdk1.4.0_02\bin;
   ```
7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:
   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```
8. Copy the PortDescription.java and ShowDescriptionWindow.java files to a folder on your computer. On the command prompt, use the cd command to move to that folder where you have copied the Java files. Next, compile the files using the javac command as follows:
   ```
   javac *.java
   ```
9. To run the Port Information application, specify the following command at the command prompt:
   ```
   java PortDescription
   ```
10. Click the List Serial Ports button on the Port Information application window to list all the serial ports attached to a computer. The Serial Ports list box displays the serial ports, as shown in Figure 2-3:

Figure 2-3: Displaying the List of Serial Ports

11. Click the List Parallel Ports button to list the parallel ports attached to a computer. The Parallel Ports list box displays a list of parallel ports, as shown in Figure 2-4:

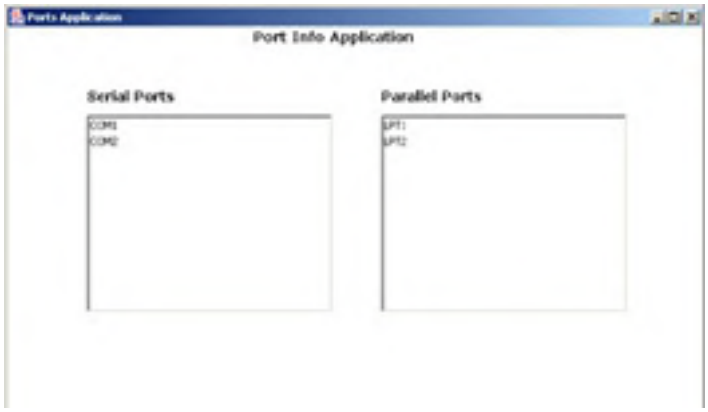

Figure 2-4: Displaying the List of Parallel Ports

12. Select a port from the list of ports and click the Port Description button. The description of the selected port appears, as shown in Figure 2-5:

Figure 2-5: Displaying the Port Description

13. Click the OK button to close the Port Description window.
14. Select a port from the list of ports and click the Open Port button. The selected port opens and a confirmation message appears, as shown in Figure 2-6:

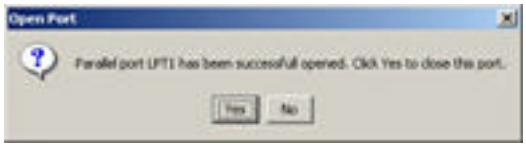

Figure 2-6: The Open Port Window

15. Click the Yes button to close the port. If you clicked the No button and again try to open the same port. The error message appears, as shown in Figure 2-7:

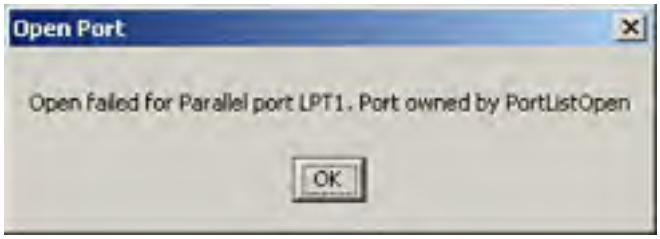

Figure 2-7: An Error Message

PREVIOUS NEXT

Chapter 3: Creating the Modem Dialer Application

The Java Communication Application Programming Interface (JCA) supports the javax.comm package. This package provides the CommPort and ParallelPort classes to obtain a list of the ports on a computer. In addition, the javax.comm package enables you to use the DataInputStream and PrintStream classes of the java.io package to send data to and receive data from the Serial port.

This chapter explains how to develop a Modem Dialer application that uses javax.comm to communicate with a modem attached to a computer and make a phone call to a specified number.

Architecture of the Modem Dialer Application

The Modem Dialer application lists all the serial and parallel ports of a computer. The application also allows an end user to select a port to which the modem is connected and make a phone call to a specified number using the modem.

> **Note** The Modem Dialer application created in this chapter uses the Serial port for the modem.

The Modem Dialer application uses the following files:

- PortModem.java: Creates the user interface of the Modem Dialer application. The end user can use this user interface to specify a phone number.
- PortChooser.java: Creates a user interface in which an end user can select the port to which the modem is connected.

Figure 3-1 shows the architecture of the Modem Dialer application:

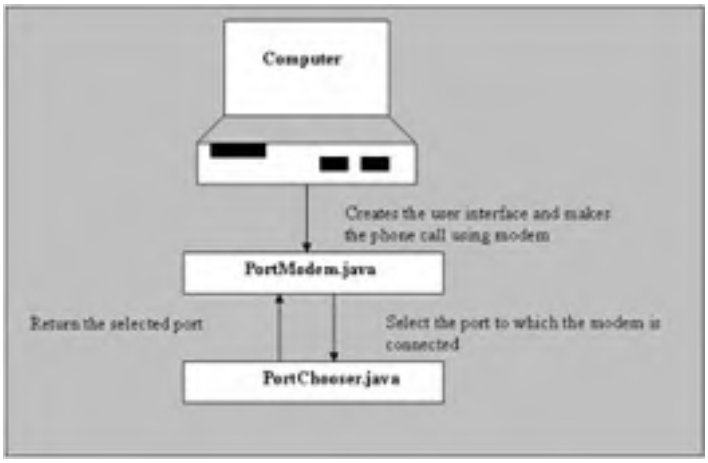

Figure 3-1: The Architecture of the Modem Dialer Application

The PortModem.java file invokes the PortChooser.java file when the end user selects the Browse button to select the port to which the modem is connected from the user interface of the Modem Dialer application.

Creating the User Interface

The PortModem.java file creates the user interface of the Modem Dialer application. Listing 3-1 shows the content of the PortModem.java file:

Listing 3-1: The PortModem.java File

```
/*Imports required Comm package classes.*/
import javax.comm.*;
/*Imports required I/O package classes.*/
import java.io.*;
/*Imports required java.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required AWT classes.*/
import java.awt.*;
import java.awt.event.*;
/*
class PortModem - This class is the main class of the application.
This class initializes the interface and loads all components like the button,
textfields before displaying the result.
Constructor:
   PortModem-This constructor creates GUI.
Methods:
   expect - Get response from modem
   send - send commands to modem
main - This method creates the main window of the application and displays all the components.
*/
public class PortModem extends JFrame implements ActionListener
{
   /*Declare objects of JButton class.*/
   private JButton dialbutton;
   private JButton cancelbutton;
   private JButton closebutton;
   private JButton portbrowsebutton;
   /*Declare objects of JPanel class.*/
   private JPanel bottompanellower;
   private JPanel bottompanelupper;
   private JPanel middlepanel;
   private JPanel bottompanel;
   /*Declare objects of JLabel class.*/
   private JLabel applicationtitlelabel;
   private JLabel portnamelabel;
   private JLabel phonenumberlabel;
   private JLabel processlabel;
   /*Declare objects of JTextField class.*/
   private JTextField portnametextbox;
   private JTextField phonenumbertextbox;
   /*Declare constraints.*/
   private final int BUTTON_WIDTH=80;
   private final int BUTTON_HEIGHT=23;
   private final int TEXTBOX_WIDTH=150;
   public static final int BAUD = 9600;
   protected boolean start = true;
   /*Declare object of DataInputStream class.*/
   protected DataInputStream is;
   /*Declare object of PrintStream class.*/
   protected PrintStream os;
   String response;
   /*
   Declare and initialize object of CommPortIdentifier class.
   */
   CommPortIdentifier commportid=null;
   /*
   Declare and initialize object of CommPort class.
   */
   CommPort commport=null;
   /*Declare object of PortChooser class.*/
   PortChooser portchooser;
   public PortModem() throws IOException, NoSuchPortException,PortInUseException,UnsupportedCommOper
   {
      /*Set parent window title.*/
      super("Modem Dialer Application");
      /*
      Initialize and set the Look and Feel of the application to Windows Look and Feel.
      */
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         System.out.println("Unable to set WLF"+e);
      }
```

```
/*Override windowClosing method.*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
      System.exit(0);
   }
});
/*
Declare and initialize object of Container class.
*/
Container contentpane=getContentPane();
/*Set background color to white.*/
contentpane.setBackground(Color.white);
/*
Declare and initialize object of Dimension class.
*/
Dimension  screendimension=Toolkit.getDefaultToolkit().getScreenSize();
/*Set window's location on screen.*/
setLocation(screendimension.width/2- 200,screendimension.height/2-200);
/*Initialize object of JLabel.*/
applicationtitlelabel=new JLabel("Phone Call Application",JLabel.CENTER);
/*Set label font.*/
applicationtitlelabel.setFont(new Font("Verdana",Font.BOLD,14));
/*Add label to contentpane.*/
contentpane.add(applicationtitlelabel,BorderLayout.NORTH);
/*Initialize object of JButton.*/
dialbutton=new JButton("Dial");
/*Set button size.*/
dialbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
dialbutton.setActionCommand("dial");
/*Add action listener to button.*/
dialbutton.addActionListener(this);
/*Set mnemonic for button .*/
dialbutton.setMnemonic('D');
dialbutton.setToolTipText("Click this button for dialing.");
/*Initialize object of JButton.*/
cancelbutton=new JButton("Cancel");
cancelbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
cancelbutton.setEnabled(false);
cancelbutton.setActionCommand("cancel");
/*Add action listener to button.*/
cancelbutton.addActionListener(this);
/*Set mnemonic for button.*/
cancelbutton.setMnemonic('C');
/*Set tool tip text.*/
cancelbutton.setToolTipText("Click this button for cancel dialing.");
/*Initialize object of Jbutton.*/
closebutton=new JButton("Close");
/*Set button size.*/
closebutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
closebutton.setActionCommand("close");
/*Add action listener to button.*/
closebutton.addActionListener(this);
/*Set mnemonic for button.*/
closebutton.setMnemonic('l');
/*Add tooltip text with button.*/
closebutton.setToolTipText("Click this button to close this window.");
/*Initialize object of JPanel.*/
bottompanellower=new JPanel();
/*Add buttons to panel.*/
bottompanellower.add(dialbutton);
bottompanellower.add(cancelbutton);
bottompanellower.add(closebutton);
/*Initialize object of JPanel.*/
bottompanelupper=new JPanel();
/*Initialize object of JLabel.*/
processlabel=new JLabel("Disconnected");
processlabel.setForeground(Color.blue);
/*Add label to panel.*/
bottompanelupper.add(processlabel);
/*
Initialize object of JPanel and set the layout as GridLayout.
*/
bottompanel=new JPanel(new GridLayout(2,1));
/*Add panels to bottompanel.*/
bottompanel.add(bottompanelupper);
bottompanel.add(bottompanellower);
/*Add bottompanel pane to contentpane.*/
contentpane.add(bottompanel,BorderLayout.SOUTH);
/*
Declare and initialize object of GridBagLayout.
*/
GridBagLayout gridlayout=new GridBagLayout();
```

```
/*
Declare and initialize object of GridBagConstraints.
*/
GridBagConstraints gridbagconstraints =new GridBagConstraints();
/*
Initialize object of JPanel class and set layout as GridBagLayout.
*/
middlepanel=new JPanel(gridlayout);
/*initialize object of JLabel.*/
portnamelabel=new JLabel("Port Name",JLabel.LEFT) ;
/*Set constraints for portnamelabel.*/
gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraints.anchor=GridBagConstraints.CENTER;
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridlayout.setConstraints(portnamelabel,gridbagconstraints);
/*Add label to pane.*/
middlepanel.add(portnamelabel);
/*Initialize object of JtextField.*/
portnametextbox=new JTextField();
/*Set button size.*/
portnametextbox.setPreferredSize(new Dimension(TEXTBOX_WIDTH,BUTTON_HEIGHT));
/*Set constraints for portnametextbox.*/
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
portnametextbox.setEnabled(false);
gridlayout.setConstraints(portnametextbox,gridbagconstraints);
/*Add label to pane.*/
middlepanel.add(portnametextbox);
/*Initialize object of JButton.*/
portbrowsebutton=new JButton("Browse..");
/*Add action listener to button.*/
portbrowsebutton.addActionListener(this);
/*Add action command to button.*/
portbrowsebutton.setActionCommand("portbrowse");
/*Set button size.*/
portbrowsebutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Set constraints for portbrowsebutton.*/
gridbagconstraints.gridx=2;
gridbagconstraints.gridy=0;
gridlayout.setConstraints(portbrowsebutton,gridbagconstraints);
middlepanel.add(portbrowsebutton);
/*
Initialize object of JLabel and set its title.
*/
phonenumberlabel=new JLabel("Phone Number",JLabel.LEFT);
/*Set constraints for phonenumberlabel.*/
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=1;
gridbagconstraints.insets=new Insets(5,0,0,0);
gridlayout.setConstraints(phonenumberlabel,gridbagconstraints);
middlepanel.add(phonenumberlabel);
/*Initialize object of JTextField.*/
phonenumbertextbox=new JTextField();
/*Set textbox size.*/
phonenumbertextbox.setPreferredSize(new Dimension(TEXTBOX_WIDTH,BUTTON_HEIGHT));
/*
Set constraints for phonenumbertextbox.
*/
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=1;
gridlayout.setConstraints(phonenumbertextbox,gridbagconstraints);
middlepanel.add(phonenumbertextbox);
contentpane.add(middlepanel,BorderLayout.CENTER);
/*
Initialize object as PortChooser class.
*/
portchooser=new PortChooser(this);
/*
Add action listener to ok button of portchooser object.
*/
portchooser.getOkButton().addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
      portchooser.dispose();
      if (portchooser.getSelectedName()==null||portchooser.getSelectedName()=="")
      {
      System.out.println("Select the port");
      }
      else
      {
         portnametextbox.setText(portchooser.getSelectedName());
      }
   }
});
/*Set window size.*/
setSize(400,300);
```

```
/*Make it visible.*/
setVisible(true);
}
/*
actionPerformed - This method is called when the user clicks the Dial, cancel, close button.
Parameters: ae - an ActionEvent object containing details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   /*
   This is executed when user clicks the Close button.
   */
   if("close".equals(actioncommand))
   {
      System.exit(0);
   }
   /*
   This is executed when user clicks the Browse button.
   */
   if("portbrowse".equals(actioncommand))
   {
      portchooser.setVisible(true);
   }
   /*
   This is executed when user clicks the Cancel button.
   */
   if ("cancel".equals(actioncommand))
   {
      try
      {
         /*close input and output stream.*/
         is.close();
         os.close();
         /*Port close.*/
         commport.close();
         processlabel.setText("Disconnected");
      }
      catch(IOException ie)
      {
      System.err.println("Error in input output.");
      }
   }
   /*
   This is executed when user clicks the Dial button.
   */
   if("dial".equals(actioncommand))
   {
      /*
      Declare and initialize object of String class.
      */
      String portname=portnametextbox.getText();
      if (portname==null)
      {
         JOptionPane.showMessageDialog(null," Please select one port name.","
         No port choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else if (portname.trim().length()==0)
      {
         JOptionPane.showMessageDialog(null,"Please select one port name.","
         No port choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      String phoneno=phonenumbertextbox.getText();
      if (phoneno==null)
      {
         JOptionPane.showMessageDialog(null," Please enter phone number.","
         No phone number choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else if(phoneno.trim().length()==0)
      {
         JOptionPane.showMessageDialog(null," Please enter phone number.","
         No phone number choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else
      {
         /*
         Check for phone number validation.
         */
         String phonenoaftertrim=phoneno.trim();
         String validstring="0123456789,";
         for(int i=0;i<phonenoaftertrim.length();i++)
         {
            if (validstring.indexOf(phonenoaftertrim.charAt(i))==-1)
```

```
                {
                   JOptionPane.showMessageDialog(null," Please enter a valid phone number.","
                   No phone number choosen",JOptionPane.PLAIN_MESSAGE);
                   return;
                }
             }
             /*Calls the portOpen()method.*/
             portOpen(phonenoaftertrim);
          }
       }
    }
public static void main(String[] args) throws IOException,
NoSuchPortException,PortInUseException,Unsuppor tedCommOperationException
{
   /*
   Declare and initialize object of PortModem class.
   */
   PortModem portmodem=new PortModem();
}
/*
portOpen - This method opens port.
Parameters: phonenoaftertrim - objcet of String class.
Return Value: NA
*/
public void portOpen(String phonenoaftertrim)
{
   /*Get selected port.*/

   commportid=portchooser.getSelectedIdentifier();
   processlabel.setText(" Dialing number...");
   switch (commportid.getPortType())
   {
      /*
      This will be executed if selected port is Serial.
      */
      case CommPortIdentifier.PORT_SERIAL:
      try
      {
         /*Try to open the port.*/
         commport=commportid.open("Modem Application",1000);
         commportid.addPortOwnershipListener(new OwnerShip());
      }
      catch(PortInUseException piu)
      {
         processlabel.setText("Port has not been opened.");
      }
      SerialPort serialport=(SerialPort)commport;
      try
      {
         /*
         Set parameters for serial port communication.
         */
         serialport.setSerialPortParams(BAUD, SerialPort.DATABITS_8,
         SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
      }
      catch(UnsupportedCommOperationException uco)
      {
         processlabel.setText("Port has not been opened.");
      }
      break;
      /*
      This will be executed if selected port is parallel.
      */
      case CommPortIdentifier.PORT_PARALLEL:
      try
      {
         /*Try to open the port.*/
         commport=commportid.open("Modem Applecation",1000);
         commportid.addPortOwnershipListener(new OwnerShip());
      }
      catch(PortInUseException piu)
      {
         processlabel.setText("Port has not been opened.");
      }
      ParallelPort parallelport=(ParallelPort)commport;
      break;
      default:
      throw new IllegalStateException("Unknown port type " + commportid);
   }
   try
   {
      /*Get Input stream.*/
      is = new DataInputStream(commport.getInputStream());
   }
   catch (IOException e)
   {
      System.err.println("Can't open input stream: write-only");
      is = null;
```

```
      }
      try
      {
         /*Get Output stream*/
         os = new PrintStream(commport.getOutputStream(), true);
      }
      catch(IOException ie)
      {
         System.err.println("Can't open output string.");
      }
      try
      {
         /*Calls send method.*/
         send("ATZ");
         /*Calls expect method.*/
         expect("OK");
         /*Calls send method.*/
         send("ATDT" + phonenoaftertrim);
      }
      catch(IOException ie)
      {
         System.err.println(ie);
      }
      try
      {
         is.close();
         os.close();
      }
      catch(IOException ie)
      {
         System.err.println(ie);
      }
   }
   /*
   class OwnerShip - This is a inner class. It implements CommPortOwnershipListener
   interface and override its methods.
   Methods:
   ownershipChange - Check to see ownership of a port.
   */
   class OwnerShip implements CommPortOwnershipListener
   {
      protected boolean owned = false;
      public void ownershipChange(int owner)
      {
         switch (owner)
         {
            /*
            Execute when end user clicks Dial button
            */
            case PORT_OWNED:
            System.out.println("An open succeeded.");
            owned = true;
            cancelbutton.setEnabled(true);
            dialbutton.setEnabled(false);
            break;
            /*
            Execute when end user clicks Cancel button
            */
            case PORT_UNOWNED:
            System.out.println("A close succeeded.");
            cancelbutton.setEnabled(false);
            dialbutton.setEnabled(true);
            owned = false;
            break;
            case PORT_OWNERSHIP_REQUESTED:
            if (owned)
            {
               if (JOptionPane.showConfirmDialog(null,"I've been asked to give up the port,
               should I?","",JOptionPane.OK_CANCEL_OPTION) == 0)
               commport.close();
            }
            else
            {
            System.out.println("Somebody else has the port");
            }
         }
      }
   }
   /*
   send - This method sends phone number to modem.
   Parameters: s - object of String class.
   Return Value: NA
   */
   protected void send(String s) throws IOException
   {
      if (start)
      {
         System.out.print(">>> ");
```

```
            System.out.print(s);
            System.out.println();
         }
         os.print(s);
         os.print("\r\n");
         if (!expect(s))
         {
            System.err.println("WARNING: Modem did not echo command.");
         }
      }
      /*
      expect - This method receives response of modem
      Parameters:   exp - objcet of String class.
      Return Value: NA
      */
      protected boolean expect(String exp) throws IOException
      {
         response = is.readLine();
         if (start)
         {
            System.out.print(response);
            System.out.println();
         }
         return response.indexOf(exp) >= 0;
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortModem class, which generates the main window of the Modem Dialer application, as shown in Figure 3-2:

Figure 3-2: The Modem Dialer Application Window

In the user interface:

- The Port Name text box: Specifies the port to which the modem is connected.
- The Phone Number text box: Specifies the phone number to which an end user wants to connect.
- The Browse button: Invokes the user interface that the PortChooser.java file creates.
- The Dial button: Enables an end user to make a phone call to the number specified in the Phone Number text box.
- The Cancel button: Ends the current call.
- The Close button: Terminates the Modem Dialer application.

The PortModem class contains an inner class called OwnerShip. The OwnerShip class implements the CommPortOwnershipListener interface provided by the JCA API and overrides the methods of the CommPortOwnershipListener interface. The various methods defined in the PortModem class are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the button that the end user clicks. When the end user clicks the Dial button, the actionPerformed() method validates the phone number specified in the Phone Number text box. In addition, the actionPerformed() method verifies the port selected in the Port Name text box. The actionPerformed() method calls the portOpen() method to open the selected port.
- `send()`: Sends the phone number specified by the end user to the modem.
- `expect()`: Receives the response of the modem through the Serial port.
- `portOpen()`: Opens the port that the end user selects from the Port Name text box. The portOpen() method creates an instance of the DataInputStream class using the input stream of the Serial port opened for communication. The DataInputStream class sends data to the modem with the help of the Serial port. The portOpen() method also creates an instance of the PrintStream class using the output stream of the Serial port opened for communication. The PrintStream class accepts the response of the modem with the help of the Serial port.

The portOpen() method also invokes the send() and expect() methods to communicate with the Serial port. The portOpen() method invokes the send() method with the ATZ parameter to check if the modem is ready for communication. If the modem responds successfully, the portOpen() method invokes the send() method with two parameters, ATDT and the phone number specified by the end user. The ATDT parameter commands the modem to dial a specified phone number.

◄ PREVIOUS NEXT ►

Selecting a Port

The PortChooser.java file creates and displays a user interface to select the port to which the modem is connected. The modem makes a phone call to the phone number specified by the end user.

Listing 3-2 shows the PortChooser.java file:

Listing 3-2: The PortChooser.java File

```
/*Imports required Comm package classes.*/
import javax.comm.*;
/*Imports required I/O package classes.*/
import java.io.*;
/*Imports required AWT classes.*/
import java.awt.*;
import java.awt.event.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
/*Imports required java.util package classes.*/
import java.util.*;
/*
class PortChooser - This class is the main class of the application.
This class initializes the interface and loads all components like the button,
combobox before displaying the result.
Constructor:
PortChooser - This constructor calls methods to create GUI and populating port list.
Methods:
   getSelectedName - Gives comm port name.
   getSelectedIdentifier - Gives comm port.
   makeGUI - Creates GUI.
populate - Generates list of serial and parallel ports
itemStateChanged: This method is called when end user select item from combobox.
*/
public class PortChooser extends JFrame implements ItemListener
{
   /*
   Declare and initialize object of HashMap class.
   */
   protected HashMap map = new HashMap();
   /*
   Declare and initialize object of CommPortIdentifier class.
   */
   protected CommPortIdentifier selectedportidentifier=null;
   /* Declare objects of JComboBox class.*/
   protected JComboBox serialportschoice;
   protected JComboBox parallelportschoice;
   /* Declare objects of JButton class.*/
   private JButton okbutton;
   private JButton cancelbutton;
   /* Declare objects of JPanel class.*/
   private JPanel bottompane;
   private JPanel centerpanel;
   /*Declare constraints.*/
   protected String selectedportname=null;
   protected final int PAD =15;
   private final int BUTTON_WIDTH=80;
   private final int BUTTON_HEIGHT=23;
   private boolean itemchangeflag=true;
   /*
   getSelectedName: It is called to get selected port name
   Parameter: NA
   Return Value: String
   */
   public String getSelectedName()
   {
      return selectedportname;
   }
   /*
   getSelectedIdentifier: It is called to get selected port.
   Parameter: NA
   Return Value: CommPortIdentifier
   */
   public CommPortIdentifier getSelectedIdentifier()
   {
      return selectedportidentifier;
   }
   public PortChooser(JFrame parent)
   {
      /*Set window title.*/
      super(" Selecting Port ");
      /*calls makeGUI method.*/
      makeGUI(parent);
      /*calls populate method.*/
      populate();
```

```
}
/*
makeGUI: This method creates GUI.
Parameter: parent - Object of JFrame.
Return Value: NA
*/
public void makeGUI(JFrame parent)
{
   /*
   Declare and initialize object of Container class.
   */
   Container cp = getContentPane();
   /*Set window's location on screen.*/
setLocation(parent.getLocation().x,parent.getLocation().y);
   /*Initialize object of JPanel.*/
   centerpanel = new JPanel();
   /*Add centerpanel to ContentPane.*/
   cp.add(BorderLayout.CENTER, centerpanel);
   /*Set layout as GridLayout.*/
   centerpanel.setLayout(new GridLayout(0, 2, PAD, PAD));
   /*
   Add object of JLabel class to ContentPane.
   */
   centerpanel.add(new JLabel("Serial Ports", JLabel.RIGHT));
   /*
   Initialize object of JComboBox class.
   */
   serialportschoice = new JComboBox();
   /*
   Add action listener to serialportschoice Combobox.
   */
   serialportschoice.addItemListener(this);
   /*
   Add serialportschoice Combobox to centerpanel panel.
   */
   centerpanel.add(serialportschoice);
   /*
   Add object of JLabel class to ContentPane.
   */
   centerpanel.add(new JLabel("Parallel Ports", JLabel.RIGHT));
   /*Initialize object of JComboBox class.*/
   parallelportschoice = new JComboBox();
   /*
   Add action listener to parallelportschoice Combobox.
   */
parallelportschoice.addItemListener(this);
   /*
   Add serialportschoice Combobox to centerpanel panel.
   */
   centerpanel.add(parallelportschoice);
   /*Initialize object of JPanel class*/
   bottompane=new JPanel();
   /*
   Initialize objects of JButton class
   */
   okbutton = new JButton("OK");
   cancelbutton = new JButton("Cancel");
   /*Set buttons size.*/
   okbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
   cancelbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
   /*Add buttons to bottompane panel.*/
   bottompane.add(okbutton);
   bottompane.add(cancelbutton);
   /*
   Add bottompane panel to cp ContentPane.
   */
   cp.add(bottompane,BorderLayout.SOUTH);
   /* Set size of window.*/
   setSize(280,120);
   /*Add actionlistener to cancelbutton.*/
   cancelbutton.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {
      PortChooser.this.dispose();
      }
   });
}
/*
populate: This method generates ports list.
Parameter: NA
Return Value: NA
*/
public void populate()
{
   serialportschoice.addItem("");
   parallelportschoice.addItem("");
```

```
      /*
Declare object of Enumeration class and fills it with the ports obtains
by getPortIdentifiers method of CommPortIdentifier.
      */
      Enumeration pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
   CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
         /*
         Put comm port in to object of hashmap class.
         */
         map.put(cpi.getName(), cpi);
         /*
         This will execute if port is serial.
         */
         if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
   serialportschoice.addItem(cpi.getName());
         }
         /*
         This will execute if port is parallel.
         */
         else if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
         {
   parallelportschoice.addItem(cpi.getName());
         }
      }
      serialportschoice.setSelectedIndex(0);
      parallelportschoice.setSelectedIndex(0);
   }
   /*
   itemStateChanged: This method is called when end user select item from combobox.
   Parameter: e - object of ItemEvent class.
   Return Value: NA
   */
   public void itemStateChanged(ItemEvent e)
   {
   if ((String)((JComboBox)e.getSource()).getSelectedItem()!="")
      {
         /*Get selected item name. */
   selectedportname = (String)((JComboBox)e.getSource()).getSelectedItem();
   selectedportidentifier = (CommPortIdentifier)map.get(selectedportname);
   if (serialportschoice.equals((JComboBox)e.getSource()))
         {
   if (parallelportschoice.getSelectedIndex()>0)
            {
   parallelportschoice.setSelectedIndex(0);
            }
         }
         else
         {
   serialportschoice.setSelectedIndex(0);
         }
      }
   }
   /*
   getOkButton: This method returns an object of OK button.
   Parameter: NA
   Return Value: JButton
   */
   public JButton getOkButton()
   {
      return okbutton;
   }
}
```

Download this listing.

The above listing displays a user interface that lists the ports on a computer. From this list, the end user can select the port to which the modem is connected. The PortChooser.java file uses the CommPortIdentifier class of javax.comm to retrieve the list of ports on a computer.

The user interface of the PortChooser.java file appears when an end user clicks Browse in the user interface of the Modem Dialer application. Figure 3-3 shows the user interface of the PortChooser.java file:

Figure 3-3: The User Interface of the PortChooser.java File

In the PortChooser.java file, the default constructor of the PortChooser class accepts an object of the PortModem class as an input parameter. The various methods defined in the PortChooser class are:

- `makeGUI()`: Creates the user interface to select the Serial port to which the modem is connected.
- `Populate()`: Generates a list of serial and parallel ports.
- `itemStateChanged()`: Retrieves the item that is currently selected in the Serial Ports or Parallel Ports drop-down list box.
- `getSelectedName()`: Returns the name of the selected Serial port.
- `getSelectedIdentifier()`: Returns an object of the CommPortIdentifier class that represents the selected Serial port.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the button that the end user clicks. When the end user clicks the OK button, the actionPerformed() method opens the selected port . If the end user clicks the Cancel button, the actionPerformed() method closes the user interface of the PortChooser.java file.

PREVIOUS NEXT

Unit Testing

To test the Modem Dialer application:

1. Download the Javacomm20-win32 JCA, which is available in zip format. The JCA is available at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the PortModem.java and PortChooser.java files to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Run the Modem Dialer application using the following command at the command prompt:

   ```
   java PortModem
   ```

10. Click the Browse button in the Modem Dialer window. The user interface of the PortChooser.java file opens, as shown in Figure 3-3.
11. Select the port to which the modem is connected and click the OK button. This opens the port selected by the end user to make the phone call.
12. Specify the phone number to which the call is to be made in the Phone Number text box. Figure 3-4 shows the user interface of the Modem Dialer application with the specified port and phone number:

Figure 3-4: Making a Phone Call

13. Click the Dial button. This makes a call to the number specified in the Phone Number text box.
14. Click the Cancel button to end the call.
15. Click the Close button to terminate the Modem Dialer application.

Chapter 4: Creating a Printing Application

The Java Communication API (JCA) supports the javax.comm package. This package enables you to use the CommPortIdentifier and CommPort classes to detect the printer port. The CommPortIdentifier class provides a list of the ports available on your computer and the CommPort class opens a selected port for input/output operations.

This chapter explains how to develop a Printing application that uses the javax.comm package to select the printer port and print the text in a document. In addition, the application allows an end user to select and change the font and the color of the text in the document.

Architecture of the Printing Application

An end user can use the Printing application to list the ports on a computer. The end user can select the printer port, and open and print any text document. The end user can also change the font and the color of selected text in the document.

The Port Communication application uses the following files:

- CommPrinting.java: Creates the user interface of the Printing application. This is the main file of the application.
- ColorChoose.java: Enables the end user to use the Printing application to select the color of the text in the document to be printed.
- FontChoose.java: Enables the end user to use the Printing application to select the font of the text in the document to be printed.
- PortChoice.java: Lists the ports on a computer and enables the end user to select the port to which the printer is connected.
- PrintComponent_Class.java: Specifies print setup properties and prints the text in the document that is currently open in the Printing application.

Figure 4-1 shows the architecture of the Printing application:

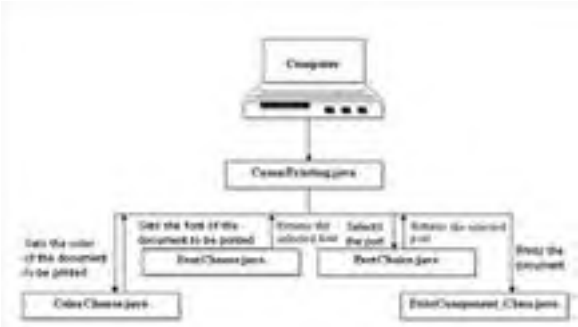

Figure 4-1: Architecture of the Printing Application

In the Printing application, the CommPrinting.java file displays a user interface that consists of a menu bar and an edit pane. The various menu options on the menu bar enable an end user to specify the color and the font of the text in the document to be printed.

The CommPrinting.java file invokes:

- The ColorChoose.java file when the end user selects the Color menu item from the Color menu.
- The FontChoose.java file when the end user selects the Font menu option from the Font menu of the Printing application.
- The PortChoice.java file when the end user selects the Choose COM Port to Print menu option from the File menu.
- The PrintComponent_Class file to print the text in the document using the Printing application.

Creating the User Interface

The CommPrinting.java file is the main file of the Printing application. This file displays a user interface with a menu bar and an edit pane. The menu bar contains three menus: File, Font, and Color. The menu options of these menus enable end users to open a document, select the color and the font of the text in the document, select the printer port, specify page setup, and print the text in the document.

Listing 4-1 shows the CommPrinting.java file:

Listing 4-1: The CommPrinting.java File

```
/* Imports required Comm classes */
import javax.comm.*;
/* Imports required I/O classes */
import java.io.*;
/* Imports required AWT classes */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/* Imports required Filedialogbox classes */
import javax.swing.filechooser.*;
/* Imports required Print classes */
import javax.print.*;
import javax.print.attribute.*;
import javax.print.event.*;
import java.awt.print.*;
import java.lang.reflect.*;
import java.awt.Graphics2D;
import javax.print.attribute.standard.*;
/*
class CommPrinting - This class is the main class of the application. This class initializes
the interface and loads all components like the menubar,editpane before displaying the result.
   Constructor:
CommPrinting-This constructor creates GUI.
   Methods:
         createGUI-This method creates GUI.
addlEvent - This method add event to components
actionPerformed - This method describes which action will be performed on which component.
getExtension - This method gives file extension.
printDoc - This method print document.
main - This method creates the main window of the application and displays all the components.
*/
class CommPrinting extends JFrame  implements ActionListener
{
   /*
      Declare object of JMenuBar class.
   */
   JMenuBar menubar;
   /*
      Declare objects of JMenu class.
   */
    JMenu file_menu;
   JMenu setting_menu;
   /*
      Declare objects of JMenuItem class.
   */
JMenuItem print_menuitem, setup_menuitem,  exit_menuitem, open_menuitem,font_menuitem,color_menuitem
   /*
      Declare object of JFileChooser class.
   */
   JFileChooser filechooser;
   /*
      Declare object of JScrollPane class.
   */
   JScrollPane scrollpane;
   /*
      Declare object of JTextPane class.
   */
   JTextPane textpane;
   /*
      Declare object of DataInputStream class.
   */
   protected DataInputStream is;
   /*
      Declare object of PrintStream class.
   */
   protected PrintStream os;
   /*
      Declare object of InputStream class.
   */
   protected InputStream istr;
   /*
      Declare object of CommPort class.
   */
```

```
CommPort commport;
/*
Declare and Initialize object of FontChoose class.
*/
public FontChoose font = new FontChoose();
/*
Declare and Initialize object of ColorChoose class.
*/
public ColorChoose color=new  ColorChoose();
/*
Declare and Initialize object of PortChoice class.
*/
PortChoice comport=new PortChoice(null,this);
private boolean PrintJobDone = false;
public static final int TIMEOUTSECONDS = 30;
public static final int BAUD = 9600;
/*
Main method that creates the instance of the CommPrinting class.
*/
public static void main(String[] args) throws IOException, NoSuchPortException, _
PortInUseException, UnsupportedCommOperationException
{
   CommPrinting commprint=new CommPrinting();
}
public CommPrinting()
{
   super("Printing Application");
   createGUI();
   addlEvent();
}
/*
   createGUI: It is called to create menubar and textpane.
   Parameter: N/A
   Return Value: N/A
*/
public void createGUI()
{
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   Container container=getContentPane();
   /*
   Set the layout of the frame as BorderLayout.
   */
   container.setLayout(new BorderLayout());
   /*
   Initialize the object of JMenuBar class.
   */
   menubar = new JMenuBar();
   /*
   Initialize the objects of JMenu class and set the Title.
   */
        file_menu = new JMenu("File");
   setting_menu=new JMenu("Settings");
   /*
   Initialize the objects of JMenuItem class and set the Title.
   */
        open_menuitem = new JMenuItem("Open");
         setup_menuitem = new JMenuItem("Page Setup & Print");
         print_menuitem = new JMenuItem("Choose COM Port to Print");
        exit_menuitem = new JMenuItem("Exit");
        font_menuitem = new JMenuItem("Font");
   color_menuitem=new JMenuItem("Color");
   /*
   Initialize the object of JTextPane class.
   */
   textpane=new JTextPane();
   /*
   Initialize the object of JScrollPane class and set this on textpane object.
   */
   scrollpane=new JScrollPane(textpane);
   /*
   Add scrollpane to content pane and place it in center of container.
   */
   container.add(scrollpane, BorderLayout.CENTER);
   /*
   Add menuitems to menu and set actioncommand to them.
   */
   file_menu.add(open_menuitem);
```

```
   file_menu.add(setup_menuitem);
   file_menu.add(print_menuitem);
   file_menu.add(exit_menuitem);
   setting_menu.add(font_menuitem);
   setting_menu.add(color_menuitem);
   open_menuitem.setActionCommand("Open");
   setup_menuitem.setActionCommand("Page");
   setup_menuitem.setEnabled(false);
   print_menuitem.setActionCommand("Print");
   exit_menuitem.setActionCommand("Exit");
   font_menuitem.setActionCommand("font");
   color_menuitem.setActionCommand("color");
   /*
      Add menu to menubar.
   */
   menubar.add(file_menu);
   menubar.add(setting_menu);
   setJMenuBar(menubar);
   /*
   Initialize object of JFileChooser class.
   */
   filechooser=new JFileChooser();
   /*
      Set filter with file dialog box.
   */
   filechooser.setFileFilter(new javax.swing.filechooser.FileFilter ()
  {
      public boolean accept(File f)
      {
          if (f.isDirectory())
          {
           return true;
          }
          String name = f.getName();
          if (name.endsWith(".txt"))
          {
           return true;
          }
          return false;
   }
   public String getDescription()
   {
      return ".txt";
         }
     });
   /*
   Set the location at which window will be displayed.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   /*
      Set window's size.
   */
   setSize(500,500);
   setVisible(true);
}
/*
addlEvent: It is called to add event listener with components.
      Parameter: N/A
      Return Value: N/A
*/
public void addlEvent()
{
   addWindowListener(new WindowAdapter(){public void windowClosing(WindowEvent e){System.exit(0);
   open_menuitem.addActionListener(this);
   setup_menuitem.addActionListener(this);
   print_menuitem.addActionListener(this);
   exit_menuitem.addActionListener(this);
   font_menuitem.addActionListener(this);
   color_menuitem.addActionListener(this);
}
/*
actionPerformed: It is called when user clicks open, Print setup &
print, print,Exit,font or color menu item.
Parameter: ae - an ActionEvent object containing details of the event.
Return Value: N/A
*/
 public void actionPerformed(ActionEvent ae)
 {
   /*
   This is executed when user clicks the open menuitem.
   */
   if ("Open".equals(ae.getActionCommand()))
   {
      int returnVal = filechooser.showOpenDialog(this);
      if(returnVal == JFileChooser.APPROVE_OPTION)
      {
         String fileextension=getExtension(filechooser.getSelectedFile());
```

```
            String fileabspath=filechooser.getSelectedFile().getAbsolutePath();
            if (fileextension.equals("jpg")||fileextension.equals("JPG")||
            fileextension.equals("gif")||fileextension.equals("GIF")||
            fileextension.equals("png")||fileextension.equals("PNG"))
            {
               textpane.insertIcon(new ImageIcon(fileabspath));
            }
            else
            {
               try
               {
                  FileInputStream inputStream = new
                  FileInputStream(filechooser.getSelectedFile().getAbsolutePath());
                  BufferedReader br = new BufferedReader(new InputStreamReader(inputStream));
                  StringBuffer strOut=new StringBuffer();
                  String strTemp="";
                  while ((strTemp = br.readLine())!=null)
                  {
                     strOut.append(strTemp).append("\n");
                  }
                  textpane.setText(strOut.toString());
               }
               catch(IOException e)
               {
                  System.out.println("Error in file reading"+e);
               }
               }
            }
         }
         /*
         This is executed when user clicks the Print menuitem.
         */
         if ("Print".equals(ae.getActionCommand()))
         {
            /*
            Declare object of PortChoice class.
            */
            comport.setVisible(true);
            setup_menuitem.setEnabled(true);
         }
         /*
         This is executed when user clicks the page setup and print menuitem.
         */
         if ("Page".equals(ae.getActionCommand()))
         {
            new PrintComponent_Class(textpane).pageSetupAndPrint();
         }
         /*
         This is executed when user clicks the exit menuitem.
         */
         if ("Exit".equals(ae.getActionCommand()))
         {
            System.exit(0);
         }
         /*
         This is executed when user clicks the font menuitem.
         */
         if ("font".equals(ae.getActionCommand()))
         {
            font.setVisible(true);
            font.getOk().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  textpane.setFont(font.font());
                  font.setVisible(false);
               }
            });
            font.getCancel().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  font.setVisible(false);
               }
            });
         }
         /*
         This is executed when user clicks the color menuitem.
         */
         if ("color".equals(ae.getActionCommand()))
         {
            color.pack();
            color.setVisible(true);
            color.getOk().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  textpane.setForeground(color.color());
```

```
            color.setVisible(false);
         }
      });

      color.getCancel().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            color.setVisible(false);
         }
      });
   }
}
/*
getExtension:This method will extract file extension from file.
Parameter:f - object of File class
Return Value: file extension
*/
public String getExtension(File f)
{
   if(f != null)
   {
      String filename = f.getName();
      int i = filename.lastIndexOf('.');
      if(i>0 && i<filename.length()-1)
      {
         return filename.substring(i+1).toLowerCase();
      };
   }
   return null;
}
/*
printDoc:This method will extract file extension from file.
Parameter:commportname - name of selected com port.
commid - object of CommPortIdentifier.
Return Value: NA
*/
public void printDoc(String commportname,CommPortIdentifier commid)
{
   if (commportname == null)
   {
      JOptionPane.showMessageDialog(null,"No port selected.","Port Choose",JOptionPane.PLAIN_M
   }
   else
   {
      try
      {
         commport=commid.open("DarwinSys DataComm", TIMEOUTSECONDS * 100);
      }
      catch(Exception e)
      {
      }
      ParallelPort pPort = (ParallelPort)commport;
      int mode = pPort.getMode();
      switch (mode)
      {
            case ParallelPort.LPT_MODE_ECP:
            System.out.println("Mode is: ECP");
            break;
            case ParallelPort.LPT_MODE_EPP:
            System.out.println("Mode is: EPP");
            break;
            case ParallelPort.LPT_MODE_NIBBLE:
            System.out.println("Mode is: Nibble Mode");
            break;
            case ParallelPort.LPT_MODE_PS2:
            System.out.println("Mode is: Byte mode.");
            break;
            case ParallelPort.LPT_MODE_SPP:
            System.out.println("Mode is: Compatibility mode.");
            break;
            default:
            throw new IllegalStateException
            ("Parallel mode " + mode + " invalid.");
         }
         try
         {
            is = new DataInputStream(commport.getInputStream());
            setup_menuitem.setEnabled(true);
         }
         catch (IOException e)
         {
            System.err.println();
            is = null;
         }
         try
         {
            os = new PrintStream(commport.getOutputStream(), true);
```

```
                }
                catch(IOException e)
                {
                   JOptionPane.showMessageDialog(null,"Can't open output stream.","
                   Port Choose",JOptionPane.PLAIN_MESSAGE);
                }
                int startpos=0;
                int endpos=0;
                StringBuffer srtbuffer=new StringBuffer(2500);
                srtbuffer=new StringBuffer(textpane.getText());
                endpos=srtbuffer.indexOf("\n",startpos);
                while (endpos!=-1)
                {
                   String strprint =  srtbuffer.substring(startpos,endpos) ;
                   os.println(strprint);
                   startpos=endpos+1;
                   endpos=srtbuffer.indexOf("\n",startpos);
                }
                os.close();
                commport.close();
                comport.setVisible(false);
            }
        }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the CommPrinting class. This class generates the main window of the Printing application, as shown in Figure 4-2:

Figure 4-2: The Printing Application User Interface

The edit pane enables an end user to display and edit the document to be printed.

The menu bar provides two menus, File and Settings. The File menu of the Printing application provides four menu options: Open, Page Setup & Print, Choose COM Port to Print, and Exit, as shown in Figure 4-3:

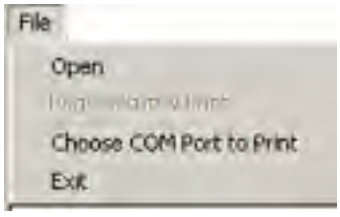

Figure 4-3: The File Menu

The Open menu option of the File menu enables the end user to open the text document to be printed. The user interface of the Printing application provides a file filter to ensure that the end user selects and opens only the text file. The Page Setup and Print menu option on the File menu specifies page setup properties and prints the text in the document. The Choose COM Port to Print menu option on the File menu allows the end user to select the COM port to which the printer is connected. The Exit menu option terminates the Printing application.

The Settings menu of the Printing application provides two menu options, Font and Color, as shown in Figure 4-4:

Figure 4-4: The Color Menu

The Font menu option enables an end user to select the font of the text in the document to be printed. The Color menu option specifies the color of the text in the document printed using the Printing application.

PREVIOUS NEXT

Selecting a Port

The PortChoice.java file allows an end user to display and select the port used to print the text in the document. The PortChoice.java file creates a user interface that displays the list of ports on a computer. The end user can select the port to which the printer is connected from the list of ports. The user interface of the PortChoice.java file appears when the end user selects the Choose COM Port to Print menu option on the File menu of the Printing application.

Listing 4-2 shows the PortChoice.java file:

Listing 4-2: The PortChoice.java File

```
/* Imports required Comm classes */
import javax.comm.*;
/* Imports required I/O classes */
import java.io.*;
/* Imports required AWT classes  */
import java.awt.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/* Imports required Util classes */
import java.util.*;
/*
class PortChoice - This class is the main class of the application. This class initializes
the interface and loads all components like the dropdown listbox, button
before displaying the result.
   Constructor:
      PortChoice-This constructor creates GUI.
   Methods:
getPortDescription - This method gives port description.
openPort - This method opens a selected port.
getSerialPorts - This method fill list of serial ports.
getCommPorts - This method fill list of comm ports.
emptyList - This method clear a list.
main - This method creates the main window of the application and displays all the components.
*/
public class PortChoice extends JDialog implements ItemListener
{
   protected String selectedPortName;
   /*
      Declare object of JButton class.
   */
   JButton okButton;
   JButton cancelButton;
   /*
      Declare object of JComboBox class.
   */
   protected JComboBox commChoice;
   /*
      Declare objects of JLabel class.
   */
   JLabel frametextlabel;
   JLabel parallelportlabel;
   /*
   Declare object of CommPortIdentifier class.
   */
   protected CommPortIdentifier selectedPortIdentifier;
   /*
      Declare object of CommPrinting class.
   */
   CommPrinting commprint;
   protected final int PAD = 5;
   /*
   itemStateChanged: This method is called where end user select item from dropdown listbox.
   Parameter: e - an ItemEvent object containing details of the event.
   Return Value: N/A
   */
   public void itemStateChanged(ItemEvent e)
   {
      selectedPortName = (String)((JComboBox)e.getSource()).getSelectedItem();
      try
      {
         selectedPortIdentifier = javax.comm.CommPortIdentifier.getPortIdentifier(selectedPortName);
      }
      catch(NoSuchPortException pe)
      {
         System.out.println("No port exists"+pe);
      }
   }
   /*
   itemStateChanged: This method is called to get selected port name.
   Parameter: N/A
```

```
Return Value: String
*/
public String getSelectedName()
{
   return selectedPortName;
}
/*
itemStateChanged: This method is called to get selected port.
Parameter: N/A
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getSelectedIdentifier()
{
   return selectedPortIdentifier;
}
public PortChoice(JFrame parent, CommPrinting cprint)
{
   super(parent, "COM Port Selection", true);
   commprint=cprint;
   makeGUI();
   populate();
   finishGUI();
}
/*
makeGUI: This method is called to create GUI.
Parameter: N/A
Return Value: N/A
*/
protected void makeGUI()
{
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Exception in setting WLAF"+e);
   }
   Container cp = getContentPane();
   setResizable(false);
   /*
   Initialize the object of JLabel class.
   */
   frametextlabel=new JLabel("Choose COM Port to Print");
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   /*
   Set the location at which window will be displayed.
   */
   setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   frametextlabel.setHorizontalAlignment(JLabel.CENTER);
   frametextlabel.setFont(new Font("Verdana", Font.BOLD, 14));
   cp.add(BorderLayout.NORTH,frametextlabel);
   /*
   Declare object of centerPanel class and initialize it.
   */
   JPanel centerPanel = new JPanel();
   cp.add(BorderLayout.CENTER, centerPanel);
   /*
      Set frame size.
   */
   setSize(250,110);
   GridBagLayout gblayout=new GridBagLayout();
   /*
      Set frame layout to GridBagLayout.
   */
   centerPanel.setLayout(gblayout);
   GridBagConstraints gbconstraints=new GridBagConstraints();
   gbconstraints.fill=GridBagConstraints.HORIZONTAL;
   gbconstraints.gridx=0;
   gbconstraints.gridy=0;
   gbconstraints.weightx=1.0;
   gbconstraints.weighty=1.0;
   gbconstraints.anchor=GridBagConstraints.CENTER;
   parallelportlabel=new JLabel("COM Ports");
   parallelportlabel.setHorizontalAlignment(JLabel.CENTER);
   parallelportlabel.setFont(new Font("Verdana", Font.PLAIN, 12));
   gblayout.setConstraints(parallelportlabel,gbconstraints);
   centerPanel.add(parallelportlabel);
   gbconstraints.gridx=1;
   gbconstraints.gridy=0;
   gbconstraints.weightx=1.0;
   gbconstraints.weighty=1.0;
   gbconstraints.gridwidth=1;
   gbconstraints.gridheight=1;
   gbconstraints.ipadx=0;
```

```
        gbconstraints.ipady=0;
        gbconstraints.anchor=GridBagConstraints.WEST;
        commChoice = new JComboBox();
        gblayout.setConstraints(commChoice,gbconstraints);
        centerPanel.add(commChoice);
        commChoice.setEnabled(false);
        addWindowListener(new WindowAdapter()
        {
           public void windowClosed(WindowEvent we)
           {
              PortChoice.this.dispose();
           }
        });
        JPanel jbuttonpanel=new JPanel(new FlowLayout());
        jbuttonpanel.add(okButton = new JButton("OK"));
        jbuttonpanel.add(cancelButton = new JButton("Cancel"));
        cp.add(BorderLayout.SOUTH, jbuttonpanel);
        okButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent e)
           {
              commprint.printDoc(getSelectedName(),getSelectedIdentifier());
           }
        });
        cancelButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              setVisible(false);
           }
        });
     }
     /*
     populate: This method is called to populate COM port in dropdown listbox.
     Parameter: N/A
     Return Value: N/A
     */
     protected void populate()
     {
        Enumeration pList = CommPortIdentifier.getPortIdentifiers();
        while (pList.hasMoreElements())
        {
           CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
           if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
           {
              commChoice.setEnabled(true);
              commChoice.addItem(cpi.getName());
           }
        }
        commChoice.setSelectedIndex(-1);
     }
     /*
     finishGUI: This method is called to add event listener to dropdown listbox
     Parameter: N/A
     Return Value: N/A
     */
     protected void finishGUI()
     {
        commChoice.addItemListener(this);
     }
}
```

Download this listing.

The above listing allows an end user to select the port used to print the text in the document. The PortChoice.java file uses the CommPortIdentifier class of the javax.comm package to retrieve the list of ports on a computer. Figure 4-5 shows the interface of the PortChoice.java file:

Figure 4-5: The Interface of the PortChoice.java File

Selecting Color

The ColorChoose.java file allows end users to select the color of the text in the document to be printed. The user interface of the ColorChoose.java file appears when an end user clicks the Color menu item from the Color menu of the Printing application.

Listing 4-3 shows the ColorChoose.java file:

Listing 4-3: The ColorChoose.java File

```
/* Import javax.swing package classes */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JSlider;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
import java.awt.Dimension;
/* Import java.awt package classes */
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Color;
/* Import javax.swing.event package classes */
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
/*
class ColorChoose - This class creates a Color dialog box that enables the
end user to change the color of the file text.
      Fields:
      panel - Contains all the components of the Color dialog
      redLabel - Contains the content of Red label
      greenLabel - Contains the content of Green label
      blueLabel - Contains the content of Blue label
      previewLabel - Contains the content of Preview label
      redSlider - Enables the end user to select the red value
      greenSlider - Enables the end user to select the green value
      blueSlider - Enables the end user to select the blue value
      labelText - Contains the content of preview text
         ok - Creates an OK button
         cancel - Creates a cancel button
         r = 0 - Stores the Red value
         g = 0 -   Stores the Green value
         b = 0 - Stores the Blue value
      Methods:
      getOK() - This method returns the OK button object
      getCancel() - This method returns the Cancel button object
      stateChanged() - This method is invoked when an end user slide slider
      color() - This method returns the color
*/
public class ColorChoose extends JDialog implements ChangeListener
{
   /* Declare the object of JPanel class */
   JPanel panel;
   /* Declare the objects of JLabel class */
   JLabel redLabel;
   JLabel greenLabel;
   JLabel blueLabel;
   JLabel previewLabel;
   /* Declare the objects of JSlider class */
   JSlider redSlider;
   JSlider greenSlider;
   JSlider blueSlider;
   /* Declare the object of JTextField class */
   JTextField labelText;
   /* Declare the objects of JButton class */
   JButton ok;
   JButton cancel;
   /* Declare the integer for storing the RGB values */
   int r = 0;
   int g = 0;
   int b = 0;
   /* Default constructor */
   public ColorChoose()
   {
      /* Set the title of the Font dialog */
      setTitle("Selecting the Color");
      /* Set resizable button to FALSE */
      setResizable(false);
      /* Initialize the object of JPanel */
      panel = new JPanel();
      /* Set the Layout as GridLayout*/
      panel.setLayout(new GridLayout(5,2,1,1));
      /* Add the panel to Color dialog frame */
```

```
        getContentPane().add(panel);
        /* Initialize and add Red label to the panel */
        redLabel = new JLabel("Red: ");
        panel.add(redLabel);
        /* Initialize and add Red slider to the panel */
        redSlider = new JSlider(0, 255, 1);
        panel.add(redSlider);
        /* Set Border to the Red slider */
        redSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Red slider */
        redSlider.addChangeListener(this);
        /* Initialize and add Green label to the panel */
        greenLabel = new JLabel("Green: ");
        panel.add(greenLabel);
        /* Initialize and add Green slider to the panel */
        greenSlider = new JSlider(0, 255, 1);
        panel.add(greenSlider);
        /* Set Border to the Green slider */
        greenSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Green slider */
        greenSlider.addChangeListener(this);
        /* Initialize and add Blue label to the panel */
        blueLabel = new JLabel("Blue: ");
        panel.add(blueLabel);
        /* Initialize and add Blue slider to the panel */
        blueSlider = new JSlider(0, 255, 1);
        panel.add(blueSlider);
        /* Set Border to the Blue slider */
        blueSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Blue slider */
        blueSlider.addChangeListener(this);
        /* Initialize and add Preview label to the panel */
        previewLabel = new JLabel("Preview: ");
        panel.add(previewLabel);
        /* Initialize and add Preview text field to the panel */
        labelText = new JTextField(10);
        panel.add(labelText);
        /* Initialize and add OK button to the panel */
        JPanel pan=new JPanel(new FlowLayout(FlowLayout.CENTER));
        ok = new JButton("OK");
        ok.setPreferredSize(new Dimension(80,23));
        pan.add(ok);
        /* Initialize and add Cancel button to the panel */
        cancel = new JButton("Cancel");
        cancel.setPreferredSize(new Dimension(80,23));
        pan.add(cancel);
        panel.add(new JLabel(" "));
        panel.add(pan);
    }
    /*
getOk() method - This method is invoked when end user click the OK button of the Color dialog box
          parameter - NA
          return value - ok
    */
    public JButton getOk()
    {
        return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel button of the Color di
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
        return cancel;
    }
    /*
    stateChanged() - This method is called when the user slides any slider of the Color dialog box.
    Parameters:   ce - an ChangeEvent object containing details of the event.
    Return Value: NA
     */
    public void stateChanged(ChangeEvent ce)
    {
        /* This section is executed, when end user slides the Red slider */
        if(ce.getSource() == redSlider)
        {
            r = redSlider.getValue();
        }
        /* This section is executed, when end user slides the Green slider */
        else if(ce.getSource() == greenSlider)
        {
            g = greenSlider.getValue();
        }
        /* This section is executed, when end user slides the Blue slider */
        else if(ce.getSource() == blueSlider)
        {
            b = blueSlider.getValue();
```

```
        }
        /* Create the object of Color class */
        Color c = new Color(r, g, b);
        /* Set the background color of the preview text field */
        labelText.setBackground(c);
    }
    /*
    color() - This method is set color of the file text
    Parameters:   NA
    Return Value: color
     */
    public Color color()
    {
        Color color = new Color(redSlider.getValue(), greenSlider.getValue(), blueSlider.getValue());
        return color;
    }
}
```

Download this listing.

The ColorChoose.java file provides a user interface with three sliders and two buttons. In addition, the user interface of the ColorChoose.java file provides a text box that shows a preview of the selected color. The end user can use the three sliders on the user interface of ColorChoose.java to specify the RGB value of the selected color. The OK button on the user interface sets the color of the text in the document to be printed to the selected color. The Cancel button on the user interface cancels the current selection. Figure 4-6 shows the user interface of the ColorChoose.java file:

Figure 4-6: The Selecting the Color Dialog Box

Clicking any button on the user interface of ColorChoose.java calls the actionPerformed() method. This method acts as a listener for the events that the end user generates.

Selecting Font

The FontChoose.java file allows an end user to select the font of the text in the document to be printed. The user interface of the FontChoose.java file appears when the end user clicks the Font menu item on the Font menu of the Printing application.

Listing 4-4 shows the FontChoose.java file:

Listing 4-4: The FontChoose.java File

```
/* Import javax.swing package classes */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Import java.awt package classes */
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/* Import javax.swing.event package classes */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.*;
/*
class FontChoose - This class creates a Font dialog box that enables the end
user to change the font family, size and type of the text.
      Fields:
      fontLabel - Contains the content of Font label
      sizeLabel - Contains the content of Size label
      typeLabel - Contains the content of Type label
      previewLabel - Contains the content of preview
      label - Contains the preview contents
      fontText - Contains the selected font family name
      typeText - Contains the selected font type name
      sizeText - Contains the selected font size name
      fontScroll - Contains the Font list
      typeScroll - Contains the Type list
      sizeScroll - Contains the Size list
      fontList - Contains all the available font family
      typeList - Contains all the available types of font style
      sizeList - Contains all the available font size
      ok - Creates an OK button
      cancel - Creates a cancel button
      Methods:
      getOK() - This method returns the OK button object
      getCancel() - This method returns the Cancel button object
      valueChanged() - This method is invoked when an end user select the item from the List box.
      font() - This method returns the font
*/
public class FontChoose extends JDialog implements ListSelectionListener
{
   /* Declare the objects of JPanel class */
   JPanel pan1;
   JPanel pan2;
   JPanel pan3;
   /* Declare the objects of JLabel class */
   JLabel fontLabel;
   JLabel sizeLabel;
   JLabel typeLabel;
   JLabel previewLabel;
   /* Declare the objects of JTextField class */
   JTextField label;
   JTextField fontText;
   JTextField sizeText;
   JTextField typeText;
   /* Declare the objects of JScrollPane class */
   JScrollPane fontScroll;
   JScrollPane typeScroll;
   JScrollPane sizeScroll;
   /* Declare the objects of JList class */
   JList fontList;
   JList sizeList;
   JList typeList;
   /* Declare the objects of JButton class */
   JButton ok;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /*
```

```
getOk() method - This method is invoked when end user click the OK button of the Font dialog box
parameter - NA
return value - ok
*/
public JButton getOk()
{
   return ok;
}
/*
getCancel() method - This method is invoked when end user click the Cancel button of the Font dia
parameter - NA
return value - cancel
*/
public JButton getCancel()
{
   return cancel;
}
/* Define the default constructor */
public FontChoose()
{
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   /* Set the title of the Font dialog */
   setTitle("Selecting the Font");
   /* Set the size of Font dialog */
   setSize(300, 400);
   /* Set resizable button to FALSE */
   setResizable(false);
   /* Initialize the object of GridBagLayout */
   gbl = new GridBagLayout();
   /* Set the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of GridBagConstraints class */
   gbc = new GridBagConstraints();
   /* Initialize the Font label object and add it to the 1,1,1,1 position with WEST alignment */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontLabel = new JLabel("Fonts: ");
   getContentPane().add(fontLabel, gbc);
   /* Initialize the Size label object and add it to the 2,1,1,1 position with WEST alignment */
   gbc.gridx = 2;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeLabel = new JLabel("Sizes: ");
   getContentPane().add(sizeLabel, gbc);
   /* Initialize the Types label object and add it to the 3,1,1,1 position with WEST alignment */
   gbc.gridx = 3;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeLabel = new JLabel("Types: ");
   getContentPane().add(typeLabel, gbc);
   /* Initialize the Font text field object and add it to the 1,2,1,1 position with WEST alignmen
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontText = new JTextField("Arial", 12);
   getContentPane().add(fontText, gbc);
   /* Initialize the Size text field object and add it to the 2,2,1,1 position with WEST alignmen
   gbc.gridx = 2;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeText = new JTextField("8", 4);
   getContentPane().add(sizeText, gbc);
   /* Initialize the Types text field object and add it to the 3,2,1,1 position with WEST alignme
   gbc.gridx = 3;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeText = new JTextField("Regular", 6);
   getContentPane().add(typeText, gbc);
   /* Initialize the Font list object and add it to the Font scroll pane object.
```

```
    Add this scroll pane object to 1,3,1,1 position with WEST alignment */
    gbc.gridx = 1;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames
    fontList = new JList(fonts);
    fontList.setFixedCellWidth(110);
    fontList.addListSelectionListener(this);
    fontList.setSelectedIndex(0);
    fontScroll = new JScrollPane(fontList);
    getContentPane().add(fontScroll, gbc);
    /* Initialize the Size list object and add it to the Size scroll pane object.
    Add this scroll pane object to 2,3,1,1 position with WEST alignment */
    gbc.gridx = 2;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] sizes = {"8", "10", "12", "14", "16", "18", "20", "24", "28", "32", "48", "72"};
    sizeList = new JList(sizes);
    sizeList.setSelectedIndex(0);
    sizeList.setFixedCellWidth(20);
    sizeList.addListSelectionListener(this);
    sizeScroll = new JScrollPane(sizeList);
    getContentPane().add(sizeScroll, gbc);
    /* Initialize the types list object and add it to the Types scroll pane object.
    Add this scroll pane object to 3,3,1,1 position with WEST alignment. */
    gbc.gridx = 3;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] types = {"Regular", "Bold", "Italic", "Bold Italic"};
    typeList = new JList(types);
    typeList.setFixedCellWidth(60);
    typeList.addListSelectionListener(this);
    typeList.setSelectedIndex(0);
    typeScroll = new JScrollPane(typeList);
    getContentPane().add(typeScroll, gbc);
    /* Initialize the preview label and add it to 1,4,3,1 position with CENTER alignment */
    gbc.gridx = 1;
    gbc.gridy = 4;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan1 = new JPanel();
    pan1.setLayout(new FlowLayout());
    previewLabel = new JLabel("Preview:");
    pan1.add(previewLabel);
    getContentPane().add(pan1, gbc);
    /* Initialize the preview text field and add it to 1,5,3,1 position with CENTER alignment */
    gbc.gridx = 1;
    gbc.gridy = 5;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan2 = new JPanel();
    pan2.setLayout(new FlowLayout());
    label = new JTextField("AaBaCcDdeEfFgGhHjJ");
    label.setEditable(false);
    label.setBorder(BorderFactory.createEtchedBorder());
    label.setFont(new Font("Verdana",Font.PLAIN,20));
    pan2.add(label);
    getContentPane().add(pan2, gbc);
    /* Initialize the OK and Cancel button. Add these two buttons to the panel. Set layout of
    the panel to FlowLayout. Now add this panel to the 1,6,4,1 position with CENTER alignment. */
    gbc.gridx = 1;
    gbc.gridy = 6;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan3 = new JPanel();
    pan3.setLayout(new FlowLayout());
    ok = new JButton("OK");
    cancel = new JButton("Cancel");
    pan3.add(ok);
    pan3.add(cancel);
    getContentPane().add(pan3, gbc);
}
/*
valueChanged() - This method is called when the user selects any manyitem from the menubar.
Parameters:   lse - a ListSelectionEvent object containing details of the event.
Return Value: NA
*/
public void valueChanged(ListSelectionEvent lse)
{
```

```
        try
        {
           /* This section is executed, when end user selects the item from Font list */
           if(lse.getSource() == fontList)
           {
           Font f1 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex(),
           Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
           fontText.setText(String.valueOf(fontList.getSelectedValue()));
           label.setFont(f1);
           }
           /* This section is executed, when end user selects the item from Size list */
           else if(lse.getSource() == sizeList)
           {
              Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
              Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
              sizeText.setText(String.valueOf(sizeList.getSelectedValue()));
              label.setFont(f2);
           }
           /* This section is executed, when end user selects the item from Type list */
           else if(lse.getSource() == typeList)
           {
              Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
              Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
              typeText.setText(String.valueOf(typeList.getSelectedValue()));
              label.setFont(f2);
           }
        }
        catch(Exception nfe)
        {}
     }
     /*
     font() - This method is set the font of the file text
     Parameters:   NA
     Return Value: font
      */
     public Font font()
     {
        /* Create an object of Font class */
        Font font = new Font(String.valueOf(fontList.getSelectedValue()), typeList.getSelectedIndex(),
        Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
        /* Return the font object */
        return font;
     }
}
```

Download this listing.

The user interface of the FontChoose.java file provides the Selecting the Font dialog box. The end user can select the style, type, and size of the font from the Selecting the Font dialog box, as shown in Figure 4-7:

Figure 4-7: The Selecting the Font Dialog Box

Clicking OK or Cancel on the user interface of the Selecting the Font dialog box invokes the actionPerformed() method. The OK button sets the font of the text in the document to be printed to the selected font. The Cancel button cancels the current selection and the control returns to the user interface of the CommPrinting.java file.

PREVIOUS NEXT

Printing the Document

The PrintComponent_Class.java file allows an end user to specify setup properties for the page and print the text in the document that is currently open in the Printing application. The PrintComponent_Class file uses the classes of the java.awt.Print package to specify and set the properties of the document to be printed.

Listing 4-5 shows the PrintComponent_Class.java file:

Listing 4-5: The PrintComponent_Class.java File

```
import java.awt.*;
import java.awt.print.*;
import java.awt.geom.*;
import javax.swing.*;
class PrintComponent_Class implements Printable
{
    private Component component;
    PrinterJob printJob = PrinterJob.getPrinterJob();
    PageFormat pageFormat= printJob.defaultPage();

    public static void printComponent(Component c)
    {
        new PrintComponent_Class(c).print();
    }
    public PrintComponent_Class(Component component)
    {
        this.component= component;
    }
    /* Set up the page and print. */
    public void pageSetupAndPrint()
    {
        pageFormat = printJob.pageDialog(pageFormat);
        printJob.setPrintable(this, pageFormat);
        if (printJob.printDialog())
        {
            try
            {
                printJob.print();
            }
            catch(PrinterException pe)
            {
               System.out.println("Error in printing !!! " + pe);
            }
        }
    }
    /* Print the page. */
    public void print()
    {
        printJob.setPrintable(this, pageFormat);
        if (printJob.printDialog())
        {
            try
            {
                printJob.print();
            }
            catch(PrinterException pe)
            {
               System.out.println("Error in printing !!! " + pe);
            }
        }
    }
    public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
    {
      Graphics2D g2 = (Graphics2D)g;
      Dimension d = component.getSize(); //get size of document
      double componentWidth = d.width; //width in pixels
      double componentHeight = d.height; //height in pixels
      double pageHeight = pf.getImageableHeight(); //height of printer page
      double pageWidth = pf.getImageableWidth(); //width of printer page
      double scale = pageWidth/componentWidth;
      int pages = (int)Math.ceil(scale * componentHeight / pageHeight);
       /* Do not print empty pages. */
       if(pageIndex >= pages)
       {
            return Printable.NO_SUCH_PAGE;
       }
      /* Shift Graphic to line up with beginning of print-imageable region */
      g2.translate(pf.getImageableX(), pf.getImageableY());
      /* shift Graphic to line up with beginning of next page to print. */
      g2.translate(0f, -pageIndex*pageHeight);
      /* Scale the page so that the width fits. */
      g2.scale(scale, scale);
      /* Repaint the page. */
      component.paint(g2);
```

```
        return Printable.PAGE_EXISTS;
    }
}
```

Download this listing.

The above listing invokes the PrintComponent_Class.java file when the end user selects the Page Setup and Print menu option. The PrintComponent_Class.java file defines the print() and pageSetupAndPrint() methods to print and specify the page setup properties of the document to be printed. PrintComponent_Class.java uses the PageFormat and PrinterJob classes of Java to print the text in the document.

PREVIOUS NEXT

Unit Testing

To test the Printing application:

1. Download the Javacomm20-win32 JCA, which is available in zip format. This JCA is available at the following URL:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java 2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the CommPrinting.java, PortChoice.java, FontChoose.java, ColorChoose.java, and Printcomponent_Class.java files to a folder on your computer. At the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Run the Printing application using the following command at the command prompt:

   ```
   java CommPrinting
   ```

10. Select File->Open to open the text document that you want to print. The selected document opens in the user interface, as shown in Figure 4-8:

Figure 4-8: The Printing Application Interface

11. Click File->Choose COM Port to Print to select the port to which the printer is connected. You can select the port to use for printing from the list of ports that appears on the user interface of PortChoice.java and click OK.
12. Select Font->Font to set the font of the text in the document to be printed. The Selecting the Font dialog box appears.
13. Select the type, style, and size of the font of the text in the document and click OK. The font of the text changes to the selected font, as shown in Figure 4-9:

Figure 4-9: Changing the Font

14. Select Color->Color to select the color of the text in the document to be printed. The Selecting the Color dialog box opens. The end user can select a specific color for the text and click OK. This changes the color of the text in the document to the selected color, as shown in Figure 4-10:

Figure 4-10: Changing the Color

15. Select File->Page Setup and Print to print the text in the document that is currently opened. This opens the Page setup dialog box, as shown in Figure 4-11:

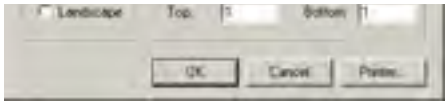

Figure 4-11: The Page Setup Dialog Box

16. Click OK. The Print dialog box opens, as shown in Figure 4-12:

Figure 4-12: The Print Dialog Box

17. Click OK.

◄ PREVIOUS NEXT ►

Chapter 5: Creating a Serial Communication Application

The Java Communication Application Programming Interface (JCA) supports the javax.comm package. This package enables you to use the CommPortIdentifier class to obtain a list of the COM or serial ports on a computer. The javax.comm package also provides the SerialPortEventListener and CommPortOwnershipListener interfaces, which include methods to handle and propagate events associated with serial or COM ports.

This chapter explains how to develop a Serial Communication application using the javax.comm package to send and receive data between the two COM or serial ports on two different computers that are attached using a null modem wire.

Architecture of the Serial Communication Application

The Serial Communication application enables two COM or serial ports to communicate with each other, which enables end users to communicate with each other. The ports of the two computers are connected through a null modem wire. The end user can use the Serial Communication application to specify the communication properties of the ports.

The Serial Communication application uses the following files:

- PortCommunication.java: Creates a user interface that helps an end user open a COM or serial port for communication, and send and receive data from the COM or serial port. This file invokes the PortPropertyPage.java file to display and set the properties of the port used for communication.
- PortPropertyPage.java: Enables the end user to specify the properties of a COM or serial port. The various properties of a COM or serial port that an end user can set are baud rate, stop bit, data bit, or parity bit.

Figure 5-1 shows the architecture of the Serial Communication application:

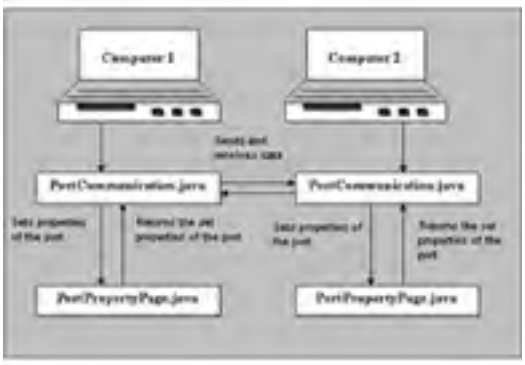

Figure 5-1: Architecture of the Serial Communication Application

Creating the User Interface and Implementing Business Logic

The PortCommunication.java file is the main file of the Serial Communication application. This file creates a user interface for the Serial Communication application. Listing 5-1 shows the PortCommunication.java file:

Listing 5-1: The PortCommunication.java File

```
import javax.comm.*;
/*Imports required java.i/o package classes.*/
import java.io.File;
import java.io.*;
/*Imports required Filedialogbox classes.*/
import javax.swing.filechooser.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Import the java.util package classes.*/
import java.util.*;
/*
class PortCommunication - This class is the main class of the application.
This class initializes the interface and loads all components like the menubar,
textareas, and buttons before displaying the result.
Constructor:
   PortCommunication-This constructor creates GUI.
Methods:
   createGUI - This method creates GUI.
   actionPerformed - This method describes which action will be performed on which component.
   getExtension - This method gives file extension.
   setConnectionParameters
   openPort - This method opens port.
   closePort - This method closes port.
   writeFile - This method write contents of textarea into file.
   save - This method save contents of textarea into file.
   main - This method creates the main window of the application and displays all the components.
*/
public class PortCommunication extends JFrame implements ActionListener,
SerialPortEventListener, CommPortOwnershipListener
{
   /*Declare object of JMenuBar class.*/
   private JMenuBar menubar;
   /* Declare object of JMenu class.*/
   private JMenu filemenu;
   /*Declare object of JMenuItem class.*/
   private JMenuItem openmenuitem, savemenuitem, exitmenuitem, property menuitem;
   /*Declare object of JTextArea class.*/
   private JTextArea sendertextarea;
   private JTextArea recevertextarea;
   /*Declare object of JScrollPane class.*/
   private JScrollPane senderscrollpane;
   private JScrollPane receverscrollpane;
   /*Declare object of JPanel class.*/
   private JPanel buttonpanel;
   private JPanel textareapanel;
   /*Declare object of JButton class.*/
   private JButton openportbutton;
   private JButton closeportbutton;
   /*Declare object of JComboBox class.*/
   private JComboBox comportcombo;
   /*
   Declare and Initialize object of File class.
   */
   File file=null;
   /*Declare object of JFileChooser class.*/
   JFileChooser openfile;
   /* Declare object of OutputStream class.*/
   private OutputStream os;
   /*Declare object of InputStream class.*/
   private InputStream is;
   /*Declare object of KeyHandler class.*/
   private KeyHandler keyHandler;
   /*Declare object of PortPropertyPage class.*/
   PortPropertyPage portpropertypage;
   /*
   Declare object of CommPortIdentifier class.
   */
   private CommPortIdentifier opencomm;
   /*Declare object of SerialPort class.*/
   private SerialPort sport;
   boolean open=false;
   private StringBuffer outputbuffer=new StringBuffer();
   public PortCommunication()
```

```
{
   super("Serial Communication");
   createGUI();
}
/*
createGUI: It is called to create menubar, text areas, and button.
   Parameter: N/A
   Return Value: N/A
*/
public void   createGUI()
{
   /*
   Set location at which window will be displayed.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   setLocation((scrnSize.width / 2) - 250, (scrnSize.height / 2) - 250);
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   /* Set window's close operation. */
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   addWindowListener(new WindowAdapter()
   {
      public void windowOpened(WindowEvent we)
      {
         System.out.println("Window opened!");
      }
   });
   /*
   Declare and Initialize object of Container class with return value of getContentPane function.
   */
   Container contentpane=getContentPane();
   /*Initialize object of JFileChooser.*/
   openfile=new JFileChooser();
   /*Set file filter for openfile object.*/
   openfile.setFileFilter(new javax.swing.filechooser.FileFilter ()
   {
   /*
   accept: Override method of FileFilter class.
   Parameter: f - Object of File class.
   Return Value:boolean
   */
   public boolean accept(File f)
   {
      if (f.isDirectory())
      {
         return true;
      }
      String name = f.getName();
      if (name.endsWith(".txt"))
      {
         return true;
      }
      return false;
   }
   /*
   getDescription: Override method of FileFilter class.
   Parameter: NA
   Return Value: String
   */
         public String getDescription()
   {
      return ".txt";
         }
        });
   /*
   Initialize the object of JMenuBar class.
   */
   menubar=new JMenuBar();
   /*
   Initialize the object of JMenu class.
   */
   filemenu=new JMenu("File");
   filemenu.setMnemonic('f');
   /*
   Initialize the object of JMenuItem class.Set hot key as 's' and add to file menu.
   */
   savemenuitem=new JMenuItem("Save");
   savemenuitem.addActionListener(this);
   savemenuitem.setMnemonic('s');
```

```
savemenuitem.setActionCommand("save");
filemenu.add(savemenuitem);
/*
Initialize the object of JMenuItem class. Set hot key as 'o' and add to file menu.
*/
propertymenuitem=new JMenuItem("Properties");
propertymenuitem.addActionListener(this);
propertymenuitem.setActionCommand("open_property");
propertymenuitem.setMnemonic('o');
filemenu.add(propertymenuitem);
/*
Initialize the object of JMenuItem class.Set hot key as 'e' and add to file menu.
*/
exitmenuitem=new JMenuItem("Exit");
exitmenuitem.addActionListener(this);
exitmenuitem.setActionCommand("exit");
exitmenuitem.setMnemonic('e');
filemenu.add(exitmenuitem);
/*
Initialize the object of JMenu class.
*/
menubar.add(filemenu);
setJMenuBar(menubar);
/* Initialize object of JPanel.*/
buttonpanel=new JPanel();
/*
Initialize object of JButton, set title, hotkey and action command of this button
*/
openportbutton=new JButton("<html>O<u>p</u>en Port</html>");
openportbutton.setMnemonic('p');
openportbutton.setActionCommand("open_port");
openportbutton.addActionListener(this);
/*
Initialize object of JButton, set title, hotkey and action command of this button
*/
closeportbutton=new JButton("<html><u>C</u>lose Port</html>");
closeportbutton.setMnemonic('C');
closeportbutton.setEnabled(false);
closeportbutton.setForeground(Color.gray);
closeportbutton.setActionCommand("close_port");
closeportbutton.addActionListener(this);
/*
Declare and initialize objects of GridBagLayout class.
*/
GridBagLayout gridbaglayout=new GridBagLayout();
/*
Declare and initialize object of GridBagConstraints class.
*/
GridBagConstraints gridbagconstraint=new GridBagConstraints();
/*
Set buttonpanel's layout as gridbaglayout
*/
buttonpanel.setLayout(gridbaglayout);
/*
Set properties of gridbagconstraint
*/
gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(openportbutton, gridbagconstraint);
/*
Add button to buttonpanel pane
*/
buttonpanel.add(openportbutton);
/*
Set properties of gridbagconstraint
*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(closeportbutton, gridbagconstraint);
/*
Add button to buttonpanel pane
*/
buttonpanel.add(closeportbutton);
contentpane.add(buttonpanel,BorderLayout.SOUTH);
textareapanel=new JPanel(new GridLayout(2,1,5,5));
/*
Initialize objects of JTextArea class.
*/
sendertextarea=new JTextArea();
recevertextarea=new JTextArea();
/*
Set recevertextarea as non editable.
*/
```

```
        recevertextarea.setEditable(false);
        /*
        Initialize objects of JScrollPane class.
        */
        senderscrollpane=new JScrollPane(sendertextarea);
        receverscrollpane=new JScrollPane(recevertextarea);
        /*
        Add scrollPanes to textareapanel
        */
        textareapanel.add(senderscrollpane);
        textareapanel.add(receverscrollpane);
        contentpane.add(textareapanel,BorderLayout.CENTER);
        /*
        Initialize object of PortPropertyPage
        */
        portpropertypage=new PortPropertyPage();
        /*
        Add actionlistener with ok button of PortPropertyPage class.
        */
        portpropertypage.getOkButton().addActionListene r(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              if (open)
              {
                 System.out.println(portpropertypage.getCommPortName()+"--"+sport.getName());
                 if (portpropertypage.getCommPortName()==sport.getName())
                 {
                    setConnectionParameters();
                 }
                 else
                 {
                    System.out.println("Port can't be changed while another port is open");
                 }
              }
              else
              {
                 setConnectionParameters();
              }
              portpropertypage.setVisible(false);
           }
        });
        /*
        Add actionlistener with cancel button of PortPropertyPage class.
        */
        portpropertypage.getCancelButton().addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              portpropertypage.setVisible(false);
           }
        });
        /*
        Set size of main window.
        */
        setSize(400,300);
        setVisible(true);
     }
     /*
  Main method that creates the instance of the PortCommunication class.
     */
     public static void main(String args[])
     {
        PortCommunication portcommunication=new PortCommunication();
     }
     /*
     actionPerformed - This method is called when the user clicks the Open Port or
     Close Port button, selects file or property menu item.
     Parameters: ae - an ActionEvent object containing details of the event.
     Return Value: NA
      */
     public void actionPerformed(ActionEvent ae)
     {
     String componentid=ae.getActionCommand();
     /*
     This is executed when user clicks the Open port button.
     */
     if (componentid=="open_port")
     {
        if (openPort())
        {
           System.out.println("open");
           openportbutton.setEnabled(false);
           openportbutton.setForeground(Color.gray);
           closeportbutton.setEnabled(true);
           closeportbutton.setForeground(Color.black);
           System.out.println(open);
        }
```

```
    }
    /*
    This is executed when user clicks the Close port button.
    */
    if (componentid=="close_port")
    {
       if (closePort())
       {
          System.out.println("close");
          closeportbutton.setEnabled(false);
          openportbutton.setEnabled(true);
          closeportbutton.setForeground(Color.gray);
          openportbutton.setForeground(Color.black);
          System.out.println(open);
       }
    }
    /*
    This is executed when user clicks the Open property Page menu item.
    */
    if (componentid=="open_property")
    {
       portpropertypage.setVisible(true);
    }
    /*
    This is executed when user clicks the Save menu item.
    */
    if (componentid=="save")
    {
       save();
    }
    /*
    This is executed when user clicks the Exit menu Item.
    */
    if (componentid=="exit")
    {
       System.exit(0);
    }
}
    /*
closePort - This method is called to close an opened port.
Parameters: NA
Return Value:boolean
 */
public boolean closePort()
{
    if (!open)
    {
       return false ;
    }
    sendertextarea.removeKeyListener(keyHandler);
    if (sport != null)
    {
       try
       {
          /* Close the I/O streams.*/
          os.close();
          is.close();
       }
       catch (IOException e)
       {
          System.err.println(e);
       }
       /*Close serial port.*/
       sport.close();
       opencomm.removePortOwnershipListener(this);
    }
    open=false;
    return true;
}
    /*
    openPort - This method is called to open a closed port.
       Parameters: NA
       Return Value:boolean
    */
    public boolean openPort()
    {
       if (portpropertypage.getCommPortName()==null||portpropertypage.getCommPortName()=="")
       {
          JOptionPane.showMessageDialog(null," No port exists. "," Error ",JOptionPane.PLAIN_MESSAGE)
         return false;
       }
       opencomm=portpropertypage.getCommPort();
       try
       {
          sport=(SerialPort)opencomm.open("Serial Communication",2000);
          try
          {
             setConnectionParameters();
```

```
            }
            catch(Exception e)
            {
               JOptionPane.showMessageDialog(null,"  Properties can not be set for this port. ","
               Serial Communication ",JOptionPane.PLAIN_MESSAGE);
            }
         }
         catch(PortInUseException pe)
         {
            JOptionPane.showMessageDialog(null," Port is in use. "," Serial Communication ",JOptionPane
         }
         try
         {
            os = sport.getOutputStream();
            is = sport.getInputStream();
         }
         catch (IOException e)
         {
            sport.close();
            return false;
         }
         keyHandler = new KeyHandler(os);
         sendertextarea.addKeyListener(keyHandler);
         try
         {
            sport.addEventListener(this);
         }
         catch (TooManyListenersException e)
         {
            sport.close();
            return false;
         }
         sport.notifyOnDataAvailabel(true);
         sport.notifyOnBreakInterrupt(true);
         try
         {
            sport.enableReceiveTimeout(30);
         }
         catch (UnsupportedCommOperationException e)
         {
            JOptionPane.showMessageDialog(null,"Serial Communication","
            This operation is not supported by this port.",JOptionPane.PLAIN_MESSAGE);
            sport.close();
            return false;
         }
         opencomm.addPortOwnershipListener(this);
         open=true;
         return true;
      }
      /*
      setConnectionParameters - This method is called to set serial port properties.
      Parameters: NA
      Return Value:boolean
       */
      public void setConnectionParameters()
      {
         int oldbaudrate= 0;
         int olddatabits =0;
         int oldstopbits = 0;
         int oldparity = 0;
         int oldflowcontrol =0;
         if (open)
         {
             oldbaudrate = sport.getBaudRate();
             olddatabits = sport.getDataBits();
             oldstopbits = sport.getStopBits();
             oldparity = sport.getParity();
             oldflowcontrol = sport.getFlowControlMode();
            try
            {
               sport.setSerialPortParams(portpropertypage.getBaudRate(),
               portpropertypage.getDatabits(), portpropertypage.getStopbits(), _
               portpropertypage.getParity());
            }
            catch (UnsupportedCommOperationException e)
            {
               portpropertypage.setBaudRate(oldbaudrate);
               portpropertypage.setDatabits(olddatabits);
               portpropertypage.setStopbits(oldstopbits);
               portpropertypage.setParity(oldparity);
            }
            try
            {
               sport.setFlowControlMode(portpropertypage.getFlowControlIn() |
               portpropertypage.getFlowControlOut());
            }
            catch (UnsupportedCommOperationException ue)
            {
```

```
      }
   }
}
/*
getExtension: This method will extract file extension from file.
Parameter: f - object of File class
Return Value: file extension
*/
public String getExtension(File f)
{
   if(f != null)
   {
      String filename = f.getName();
      int i = filename.lastIndexOf('.');
      if(i>0 && i<filename.length()-1)
      {
         return filename.substring(i+1).toLowerCase();
      };
   }
   return null;
}
/*
save: This method will save to the contents of textarea into a file.
   Parameter- NA
   Return Value:boolean
*/
public boolean save()
{
   int result = openfile.showSaveDialog(this);
   if( result == JFileChooser.CANCEL_OPTION)
   {
      return true;
   }
   else if( result == JFileChooser.APPROVE_OPTION)
   {
      file = openfile.getSelectedFile();
      if( file.exists())
      {
         int response = JOptionPane.showConfirmDialog(null,
         "Overwrite existing file?",
         "Confirm Overwrite",
         JOptionPane.OK_CANCEL_OPTION,
         JOptionPane.QUESTION_MESSAGE);
         if( response == JOptionPane.CANCEL_OPTION)
            return false;
      }
      return writeFile(file, sendertextarea.getText());
   }
   else
   {
      return false;
   }
}
/*
writeFile: This method will save to the contents of textarea into a file.
   Parameter- file - object of File class, dataString - Object of String class.
   Return Value:boolean
*/
public static boolean writeFile(File file, String dataString)
{
   try
   {
      PrintWriter out=new PrintWriter(new BufferedWriter(new FileWriter(file)));
      out.print(dataString);
      out.flush();
      out.close();
   }
   catch (IOException e)
   {
      return false;
   }
   return true;
}
class KeyHandler extends KeyAdapter
{
   OutputStream os;
   public KeyHandler(OutputStream os)
   {
      super();
      this.os = os;
   }
   public void keyTyped(KeyEvent evt)
   {
         char newCharacter = evt.getKeyChar();
      try
      {
         if ((int)newCharacter==10)
         {
```

```
                for (int i=0;i<outputbuffer.length() ;i++ )
                {
                   os.write((int)outputbuffer.charAt(i));
                }
                outputbuffer=new StringBuffer();
             }
             else
             {
                outputbuffer.append(newCharacter);
             }
             System.out.println((int)newCharacter);
          }
          catch (IOException e)
          {
             System.err.println("OutputStream write error: " + e);
          }
       }
    }
    public void ownershipChange(int type)
    {
       if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED)
       {
       }
    }

    public void serialEvent(SerialPortEvent e)
    {
       /*
       Create a StringBuffer and int to receive input data.
       */
       StringBuffer inputBuffer = new StringBuffer();
       int newData = 0;
       /* Determine type of event.*/
       switch (e.getEventType())
       {
          case SerialPortEvent.DATA_AVAILABEL:
          while (newData != -1)
          {
             try
             {
                newData = is.read();
                if (newData == -1)
                {
                   break;
                }
                if ('\r' == (char)newData)
                {
                   inputBuffer.append('\n');
                }
                else
                {
                   inputBuffer.append((char)newData);
                }
             }
             catch (IOException ex)
             {
                System.err.println(ex);
                return;
             }
          }
          /* Append received data to messageAreaIn.*/
          recevertextarea.append(new String(inputBuffer));
             break;
          /*
          If break event append BREAK RECEIVED message.
          */
          case SerialPortEvent.BI:
             recevertextarea.append("\n--- BREAK RECEIVED ---\n");
       }
    }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortCommunication class. This class generates the main window of the Serial Communication application, as shown in Figure 5-2:

Figure 5-2: The Serial Communication Window

The user interface of the Serial Communication application displays a File menu that contains three menu options, Save, Properties, and Exit, as shown in Figure 5-3:

Figure 5-3: The File Menu

The Save menu option saves the data sent and received using the ports.

The Properties menu option enables the end user to set the properties of the port used for communication. When the end user clicks this option, it creates an instance of the PortPropertyPage.java class to open a dialog box that end users can use to change the properties of the selected port.

The Exit menu option enables the end user to exit from the application.

The user interface of the Serial Communication application also contains two text panes. The upper text pane sends data through a selected and opened port. The lower pane displays the data received using the selected and opened port.

In addition to text panes and menus, the user interface of the Serial Communication application also displays two buttons, Open Port and Close Port, to open and close the port used for communication.

When an end user clicks any button, the Serial Communication application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate method based on the button that the end user clicks. For example, when the end user clicks the Open Port button, the actionPerformed() method opens a port for communication. The actionPerformed() method invokes the setConnectionParameters() method, which returns the properties last saved by the end user. When the end user clicks the Close Port button, the actionPerformed() method closes the port that was opened for communication.

◄ PREVIOUS NEXT ►

Displaying Port Properties

The PortPropertyPage.java file allows the end user to display and set the properties of the port used to send and receive data. The various properties of the port that the PortPropertyPage.java file displays are port name, baud rate, flow control in, flow control out, data bits, stop bits, and parity bits.

Listing 5-2 shows the PortPropertyPage.java file:

Listing 5-2: The PortPropertyPage.java File

```
/*Imports required Comm classes*/
import javax.comm.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Imports required java.util package classes.*/
import java.util.*;
/*
class PortPropertyPage - This class is the Port Property dialog window,
Enable the end user to change the property of port.
Constructor:
   PortPropertyPage-This constructor creates GUI.
Methods:
   setParity- Sets Parity bit.
   setStopbits - Sets Stop bit.
   setDatabits - Set Data bit
   setBaudRate - Sets Baud rate.
   getCancelButton - Returns object of cancel button.
   getOkButton -  Returns object of ok button.
   getFlowControlOut - Returns flow control out value.
   getFlowControlIn - Returns flow control in value.
   getParity - Returns Parity bit value.
   getDatabits - Returns Data bit value.
   getStopbits - Returns Stop bit value.
   getBaudRate - Returns Baud rate value.
   getCommPortName - Returns  port name.
   getCommPort -  Returns object of CommPort class
*/
class PortPropertyPage extends JFrame   implements ActionListener
{
   /*Declare object of Enumeration class.*/
   Enumeration listport;
   /*Declare object of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /*Declare object of JPanel class.*/
   JPanel buttonpanel;
   JPanel propertylist;
   /*Declare object of JLabel class.*/
   JLabel portlistlabel;
   JLabel flowlabel;
   JLabel paritylabel;
   JLabel buadlabel;
   JLabel databitlabel;
   JLabel flowcontrollabel;
   JLabel stoplabel;
   /*Declare object of JComboBox class.*/
   JComboBox comportcombo;
   JComboBox flowcombo;
   JComboBox stopcombo;
   JComboBox databitcombo;
   JComboBox paritycombo;
   JComboBox buadcombo;
   JComboBox flowcontrolcombo;
   /*
   Declare object of CommPortIdentifier class.
   */
   CommPortIdentifier portId;
   public PortPropertyPage()
   {
      /*Set title of main window.*/
      super("Port Property Page");
      /*
      Declare and initialize object of Container class.
      */
      Container contentpane=getContentPane();
      comportcombo=new JComboBox();
      listport=CommPortIdentifier.getPortIdentifiers();
      while (listport.hasMoreElements())
      {
         portId=(CommPortIdentifier)listport.nextElement();
```

```
   if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
   {
      comportcombo.addItem(portId.getName());
   }
 }
/*Initialize Object of JComboBox.*/
flowcombo=new JComboBox();
flowcombo.addActionListener(this);
/*
Declare and initialize object of GridBagLayout class
*/
GridBagLayout gridbaglayout=new GridBagLayout();
/*
Declare and initialize object of GridBagConstraints class
*/
GridBagConstraints gridbagconstraint=new GridBagConstraints();
/*
Initialize object of JPanel class and set its layout as gridbaglayout.
*/
propertylist=new JPanel(gridbaglayout);
/*Set background color as white.*/
propertylist.setBackground(Color.white);
/*
Initialize object of JLabel class and gives its name.
*/
portlistlabel=new JLabel("Port Name",JLabel.RIGHT);
/*
Initialize object of JLabel class and gives its name.
*/
flowlabel=new JLabel("Flow Control In");
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=0;
gridbagconstraint.weightx=1.0;
gridbagconstraint.weighty=1.0;
gridbaglayout.setConstraints(portlistlabel, gridbagconstraint);
/*Add portlistlabel label to panel.*/
propertylist.add(portlistlabel);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbaglayout.setConstraints(comportcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(comportcombo);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowlabel, gridbagconstraint);
/*Add flowlabel label to panel.*/
propertylist.add(flowlabel);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(flowcombo);
/*
Initialize object of JLabel class and gives its name.
*/
databitlabel=new JLabel("Data Bits",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstrain ts.CENTER;
gridbaglayout.setConstraints(databitlabel, gridbagconstraint);
/*Add databitlabel label to panel.*/
propertylist.add(databitlabel);
/*Initialize object of JComboBox class.*/
databitcombo=new JComboBox();
databitcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(databitcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(databitcombo);
/*
Initialize object of JLabel class and gives its name.
```

```
*/
paritylabel=new JLabel("Parity Bits", JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=3;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(paritylabel, gridbagconstraint);
/*Add paritylabel label to panel.*/
propertylist.add(paritylabel);
/*Initialize object of JComboBox class.*/
paritycombo=new JComboBox();
paritycombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=3;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(paritycombo, gridbagconstraint);
/*Add paritycombo combo to panel.*/
propertylist.add(paritycombo);
/*
Initialize object of JLabel class and gives its name.
*/
buadlabel=new JLabel("Baud Rate", JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(buadlabel, gridbagconstraint);
/*Add buadlabel label to panel.*/
propertylist.add(buadlabel);
/*Initialize object of JComboBox class.*/
buadcombo=new JComboBox();
buadcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(buadcombo, gridbagconstraint);
/*Add buadcombo combo to panel.*/
propertylist.add(buadcombo);
/*
Initialize object of JLabel class and gives its name.
*/
flowcontrollabel=new JLabel("Flow Control Out",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrollabel, gridbagconstraint);
/*Add flowcontrollabel label to panel.*/
propertylist.add(flowcontrollabel);
/*Initialize object of JComboBox class.*/
flowcontrolcombo=new JComboBox();
flowcontrolcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrolcombo, gridbagconstraint);
/*Add flowcontrolcombo combo to panel.*/
propertylist.add(flowcontrolcombo);
/*
Initialize object of JLabel class and gives its name.
*/
stoplabel=new JLabel("Stop Bit",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(stoplabel, gridbagconstraint);
/*Add stoplabel label to panel.*/
propertylist.add(stoplabel);
/*Initialize object of JComboBox class.*/
stopcombo=new JComboBox();
stopcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(stopcombo, gridbagconstraint);
/*Add stopcombo combo to panel.*/
propertylist.add(stopcombo);
contentpane.add(propertylist);
/* Fill buadcombo combobox. */
buadcombo.addItem("300");
buadcombo.addItem("2400");
buadcombo.addItem("9600");
buadcombo.addItem("14400");
```

```
   buadcombo.addItem("28800");
   buadcombo.addItem("38400");
   buadcombo.addItem("57600");
   buadcombo.addItem("152000");
   buadcombo.setSelectedItem("9600");
   /*Fill flowcombo combobox.*/
   flowcombo.addItem("None");
   flowcombo.addItem("Xon/Xoff In");
   flowcombo.addItem("RTS/CTS In");
   /*Fill flowcontrolcombo combobox.*/
   flowcontrolcombo.addItem("None");
   flowcontrolcombo.addItem("Xon/Xoff Out");
   flowcontrolcombo.addItem("RTS/CTS Out");
   /*Fill stopcombo combobox.*/
   stopcombo.addItem("1");
   stopcombo.addItem("1.5");
   stopcombo.addItem("2");
   /*Fill paritycombo combobox.*/
   paritycombo.addItem("None");
   paritycombo.addItem("Even");
   paritycombo.addItem("Odd");
   /*Fill databitcombo combobox.*/
   databitcombo.addItem("5");
   databitcombo.addItem("6");
   databitcombo.addItem("7");
   databitcombo.addItem("8");
   databitcombo.setSelectedItem("8");
   /*
   Initialize object of JButton and set its label
   */
   okbutton=new JButton("OK");
   /*
   Initialize object of JButton and set its label
   */
   cancelbutton=new JButton("Cancel");
   /*Initialize object of JPanel.*/
   buttonpanel=new JPanel();
   /*
   Set buttonpanel  layout as gridbaglayout.
   */
   buttonpanel.setLayout(gridbaglayout);
   /*Set constraints for gridbagconstraint.*/
   gridbagconstraint.fill = GridBagConstraints.HORIZONTAL;
   gridbagconstraint.insets=new Insets(5, 75, 5, 5);
   gridbagconstraint.gridx=0;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(okbutton, gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(okbutton);
   /*Set constraints for gridbagconstraint.*/
   gridbagconstraint.insets=new Insets(5, 5, 5, 75);
   gridbagconstraint.gridx=1;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(cancelbutton, gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(cancelbutton);
   contentpane.add(buttonpanel,BorderLayout.SOUTH);
   addWindowListener(new WindowAdapter()
   {
      public void windowClosed(WindowEvent we)
      {
         setVisible(false);
      }
   });
   /*Set frame size.*/
   setSize(400,200);
}
public void actionPerformed(ActionEvent e)
{
   JComboBox cb = (JComboBox)e.getSource();
   String petName = (String)cb.getSelectedItem();
}
/*
getCommPort - Gives object of  CommPort class
Parameters: NA
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getCommPort()
{
   CommPortIdentifier comp=null;
   try
   {
      comp= javax.comm.CommPortIdentifier.getPortIdentifier((String)comportcombo.getSelectedItem(
   }
   catch(NoSuchPortException e)
```

```
    {
       JOptionPane.showMessageDialog(null,"open", "Port not exists", JOptionPane.PLAIN_MESSAGE);
    }
    return comp;
}
/*
getCommPortName - Gives name of selected port.
Parameters: NA
Return Value:String
*/
public String getCommPortName()
{
    return (String)comportcombo.getSelectedItem();
}
/*
getBaudRate - Gives baud rate.
Parameters: NA
Return Value:int
*/
public int getBaudRate()
{
    return Integer.parseInt((String)buadcombo.getSelectedItem());
}
/*
getDatabits - Gives Databits.
Parameters: NA
Return Value:int
*/
public int getDatabits()
{
    return Integer.parseInt((String)databitcombo.getSelectedItem());
}
/*
getStopbits - Gives Stop bit.
Parameters: NA
Return Value:int
*/
public int getStopbits()
{
    return Integer.parseInt((String)stopcombo.getSelectedItem());
}
/*
    getParity - Gives parity bit.
    Parameters: NA
    Return Value:int
*/
public int getParity()
{
    String paritybit=(String)paritycombo.getSelectedItem();
    if (paritybit.equals("None"))
    {
       return SerialPort.PARITY_NONE;
    }
    else if (paritybit.equals("Even"))
    {
       return SerialPort.PARITY_EVEN;
    }
    else if (paritybit.equals("Odd"))
    {
        return SerialPort.PARITY_ODD;
    }
    else
    {
          return SerialPort.PARITY_NONE;
    }
}
/*
getFlowControlIn - Gives type of flow control.
    Parameters: NA
    Return Value:int
*/
public int getFlowControlIn()
{
    String flowcomboin=(String)flowcombo.getSelectedItem();
    if (flowcomboin.equals("None"))
    {
       return SerialPort.FLOWCONTROL_NONE;
    }
    if (flowcomboin.equals("Xon/Xoff Out"))
    {
       return SerialPort.FLOWCONTROL_XONXOFF_OUT;
    }
    if (flowcomboin.equals("Xon/Xoff In"))
    {
       return SerialPort.FLOWCONTROL_XONXOFF_IN;
    }
    if (flowcomboin.equals("RTS/CTS In"))
    {
```

```
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowcomboin.equals("RTS/CTS Out"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }
    /*
    getFlowControlOut - Gives type of flow out control.
    Parameters: NA
    Return Value:int
    */
    public int getFlowControlOut()
    {
        String flowcontrol=(String)flowcontrolcombo.getSelectedItem();
        if (flowcontrol.equals("None"))
        {
            return SerialPort.FLOWCONTROL_NONE;
        }
        if (flowcontrol.equals("Xon/Xoff Out"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_OUT;
        }
        if (flowcontrol.equals("Xon/Xoff In"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_IN;
        }
        if (flowcontrol.equals("RTS/CTS In"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowcontrol.equals("RTS/CTS Out"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }
    /*
    getOkButton - Gives object of JButton class
    Parameters: NA
    Return Value: JButton
    */
    public JButton getOkButton()
    {
        return okbutton;
    }

    /*
    getCancelButton - Gives object of JButton class
    Parameters: NA
    Return Value: JButton
    */
    public JButton getCancelButton()
    {
        return cancelbutton;
    }
    /*
        setBaudRate - Sets value of baud rate.
        Parameters: budrate
        Return Value: NA
    */
    public void setBaudRate(int budrate)
    {
        buadcombo.setSelectedItem(String.valueOf(budrate));
    }
    /*
    setDatabits - Sets value of Data bit.
    Parameters: databit
    Return Value: NA
    */
    public void  setDatabits(int databit)
    {
    databitcombo.setSelectedItem(String.valueOf(databit));
    }
    /*
        setStopbits - Sets value of Stop bit.
        Parameters: - stopbit
        Return Value: NA
    */
    public void  setStopbits(int stopbit)
    {
    stopcombo.setSelectedItem(String.valueOf(stopbit));
    }
    /*
        setParity - Sets value of Parity bit.
        Parameters: parity
        Return Value: NA
```

```
   */
   public void  setParity(int parity)
   {
      switch(parity)
      {
         case SerialPort.PARITY_NONE:
         stopcombo.setSelectedItem("None");
         break;
         case SerialPort.PARITY_EVEN:
         stopcombo.setSelectedItem("Even");
         break;
         case SerialPort.PARITY_ODD:
         stopcombo.setSelectedItem("Odd");
         break;
         default:
         stopcombo.setSelectedItem("None");
      }
   }
}
```

Download this listing.

The above listing enables the end user to set the properties of the port. Figure 5-4 shows the interface of the PortPropertyPage.java file:

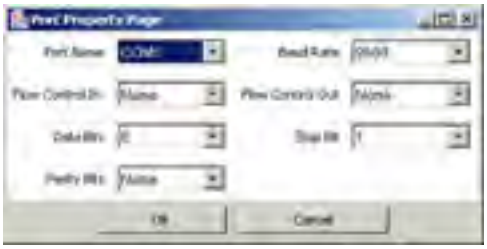

Figure 5-4: The Interface of the PortPropertyPage.java File

The user interface of the PortPropertyPage.java file displays various properties of the port. The PortPropertyPage.java file retrieves and specifies the properties of a port using the following get and set methods:

- The setParity() and getParity() methods of the PortPropertyPage class specify and retrieve the parity bit value of the port.
- The setStopbits() and getStopbits() methods of the PortPropertyPage class specify and retrieve the value of the stop bit of the port.
- The setBaudRate() and getBaudRate() methods of the PortPropertyPage class set and retrieve the baud rate property of the COM port.
- The setDatabits() and getDatabits() methods of the PortPropertyPage class specify and retrieve the value of the data bit property of the port.
- The getCancelButton() and getOKButton() methods of the PortPropertyPage class create the objects of the Cancel and OK buttons.
- The getFlowControlOut() and getFlowControlIn() methods of the PortPropertyPage class retrieve the out and in values of the flow control property.
- The getCommPorts() method of the PortPropertyPage class retrieves all the COM ports on a computer. The getCommPortName() method returns the name of the COM ports on a computer.

Unit Testing

To test the Serial Communication application:

1. Download the Javacomm20-win32 JCA, which is availabel in zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java 2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the PortCommunication.java and PortPropertyPage.java files to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Copy the PortCommunication.java and PortPropertyPage.java files to another computer on the network. Compile these files using the command specified in Step 8.
10. Connect the two computers that contain the PortCommunication.java and PortPropertyPage.java files using a null modem wire.
11. Run the Serial Communication application on both the computers connected by a null modem wire using the following command at the command prompt:

    ```
    java PortCommunication
    ```

12. Click the Open Port button in the Serial Communication window of both computers. This opens the currently selected port in the Port Property Page window for communication.
13. Click File->Properties in the Serial Communication window on both the computers connected by a null modem wire. This opens the Port Property page.
14. Select the port connected by the null modem wire in the Port Property Page window on both computers. You can set communication properties for the communicating ports in the Port Property Page window.
15. Click the OK button in the Port Property Page window to set the communication properties.
16. Type the data to be sent using the Serial Communication application. The data is specified in the upper text pane of the user interface of the Serial Communication window. Press Enter, as shown in Figure 5-5:

Figure 5-5: Sending Data Using the Serial Communication Application

The data is received in the lower pane of the Serial Communication interface on the other computer, as shown in Figure 5-6:

Figure 5-6: Receiving Data using the Serial Communication Application

PREVIOUS NEXT

Chapter 6: Creating a Fax Application

The javax.comm package includes the CommPortIdentifier class to control access to the various communication ports on a computer and the CommPort and ParallelPort classes to retrieve and set the properties of the ports on a computer that communicate with the modem and fax machine.

This chapter explains how to develop a Fax application that uses the Java Communication Application Programming Interface and the CommPort, ParallelPort, and CommPortIdentifier classes.

Architecture of the Fax Application

The Fax application allows an end user to specify a fax number, select a file, and send the content of the file as a fax to the specified number. The application sends the fax by converting a text file to an image. This process is called encoding the fax.

The Fax application uses the following files:

- PortFax.java: Creates the user interface of the Fax application.
- ConvertIntoHTML.java: Converts the text that is currently open in the edit pane of the Fax application to an image of an HTML page.
- ConvertIntoTextImage.java: Converts the text that is currently open in the edit pane of the Fax application to a text image.
- FaxPropertyPage.java: Sets the properties of the COM port that sends the fax.
- FaxStatusListener.java: Defines various static variables, such as ST_OPEN_PORT, ST_INIT_MODEM, and ST_SEND_PAGE, which indicate the status of the Fax application.
- SendingFax.java: Sends the fax to the destination.
- PreparedFaxDoc.java: Defines an interface to create the fax document. The ConvertIntoHTML.java and ConvertIntoTextImage.java files implement this interface.
- ModemCapabilities.java: Sets the encoding and decoding capabilities of a modem.
- ImageToFaxEncoder.java: Encodes the image that is sent using the Fax application.
- LCFrame.java: Converts the encoded fax image to frames. The modem sends the frames to the receiver.

Figure 6-1 shows the architecture of the Fax application:

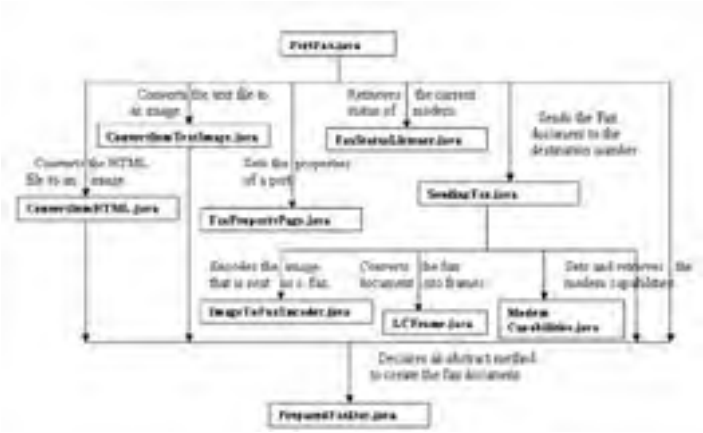

Figure 6-1: Architecture of the Fax Application

The PortFax.java file calls the ConvertIntoTextImage.java and ConvertIntoHTML.java files to convert the fax document to an image. The PortFax.java file invokes the FaxPropertyPage.java file to set the properties of the port to which the modem is connected. The FaxStatusListener.java file helps the PortFax.java file determine the status of the Fax application.

The end user sends a fax using the SendingFax.java file, which invokes the ImageToFaxEncoder.java file to encode the fax document. The LCFrame.java file helps the SendingFax.java file convert the encoded fax document into frames. The SendingFax.java file invokes the ModemCapabilities.java file to retrieve and set the encoding and decoding capabilities of a modem that helps the modem detect incoming frames to send a fax.

The PortFax.java file passes the frames to the modem, which in turn passes the frames to the fax number of the destination.

Creating the User Interface

The PortFax.java file is the main file of the Fax application. This file creates the user interface of the Fax application. The user interface of the Fax application consists of text fields to set the number to which the end user wants to send the fax, and the initial command string for the modem. In addition, the user interface of the Fax application contains an edit pane that displays the text file that the end user wants to send as a fax.

Listing 6-1 shows the PortFax.java file:

Listing 6-1: The PortFax.java File

```
/* Imports required javax.comm package classes */
import javax.comm.*;
/* Imports required java.i/o package classes */
import java.io.*;
/*Imports required java.awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required javax.swing package classes*/
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
/*
class PortFax - This class is the main class of the application. This class initializes the
interface and loads all components like the menubar, textareas, and buttons
before displaying the result.
Constructor:
   PortFax - This constructor creates GUI.
Methods:
   createGUI - This method creates GUI.
actionPerformed - This method describes which action will be performed on which component.
main - This method creates the main window of the application and displays all the components.*/

class PortFax extends JFrame implements ActionListener
{
   /*Set constants*/
   public static final int ST_OPEN_PORT = 1;
    public static final int ST_INIT_MODEM = 2;
    public static final int ST_CONNECTING = 3;
    public static final int ST_SEND_PAGE = 4;
    public static final int ST_CLOSE = 5;
    public static final int ST_CONVERT_FIlES = 6;
    public static final int ST_REC_PAGE = 4;
    public static final int ST_REC_CALL = 7;
    public static final int ST_WAIT_CALL = 8;
   /*Declare objects of String class.*/
   String oldcommname;
   String olddialing;
   String oldmodemclass;
   String oldflowcontrol;
   /*Declare objects of JLabel class.*/
   JLabel fexLabel;
   JLabel modemstringLabel;
   JLabel processlabel;
   /*Declare objects of JTextField class.*/
   JTextField modemstringtext;
   JTextField faxtextfield;
   /*Declare objects of JEditorPane class.*/
   JEditorPane faxeditor;
   /*Declare objects of JButton class.*/
   JButton sendbutton;
   JButton cancelbutton;
   JButton closebutton;
   /*Declare objects of JPanel class.*/
   JPanel toppanel;
   JPanel EditorPanel;
   /*Declare objects of JMenuBar class.*/
   JMenuBar menubar;
   /*Declare objects of JMenu class.*/
   JMenu filemenu;
   /*Declare objects of JMenuItem class.*/
   JMenuItem openmenuitem,viewpropertiesmenuitem,exitmenuitem;
   /*Declare objects of JFileChooser class.*/
   JFileChooser filechooser;
   /*Declare objects of JScrollPane class.*/
   JScrollPane editorscrollpane;
   /*Declare objects of JCheckBox class.*/
   JCheckBox htmlcheckbox;
   /*Declare objects of PreparedFaxDoc class.*/
   PreparedFaxDoc faxdoc=null;
   /*Declare objects of SendingFax class.*/
   SendingFax sendfax=null;
   /*Declare objects of FaxPropertyPage class.*/
```

```
FaxPropertyPage faxpropertypage;
int oldflowcontrolin=0;
boolean d=false;
public PortFax()
{
   /*Set parent window title.*/
   super(" Fax Application ");
   /*createGUI method is called*/
   createGUI();
   /*Set window size.*/
   setSize(500,500);
   /*Set window visible.*/
   setVisible(true);
}
public static void main(String[] args)
{
   /*Declare and initialize object of PortFax.*/
   PortFax portfax=new PortFax();
}
public void createGUI()
{
   /*
   Declare and Initialize object of Container class with return value of getContentPane function.
   */
   Container contentpane=getContentPane();
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Problem in changing windows look and feel");
   }
   /*Set location at which window will be displayed.*/
   Dimension screensize=Toolkit.getDefaultToolkit()getScreenSize();
   setLocation(screensize.width/2-250,screensize.height/250);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   /*Initialize the object of JMenuBar class.*/
   menubar=new JMenuBar();
   /*Initialize the object of JMenu class.*/
   filemenu=new JMenu("File");
   filemenu.setMnemonic('f');
   /*
   Initialize the object of JMenuItem class.
   */
   openmenuitem=new JMenuItem("Open");
   openmenuitem.setMnemonic('o');
   openmenuitem.setActionCommand("open");
   openmenuitem.addActionListener(this);
   filemenu.add(openmenuitem);
   /*
   Initialize the object of JMenuItem class.
   */
   viewpropertiesmenuitem=new JMenuItem("View Properties");
   viewpropertiesmenuitem.setMnemonic('v');
   viewpropertiesmenuitem.setActionCommand("view");
   viewpropertiesmenuitem.addActionListener(this);
   filemenu.add(viewpropertiesmenuitem);
   /*
   Initialize the object of JMenuItem class.
   */
   exitmenuitem=new JMenuItem("Exit");
   exitmenuitem.setMnemonic('e');
   exitmenuitem.setActionCommand("exit");
   exitmenuitem.addActionListener(this);
   filemenu.add(exitmenuitem);
   menubar.add(filemenu);
   setJMenuBar(menubar);
   /*
   Initialize the object of FaxPropertyPage class.
   */
   faxpropertypage=new FaxPropertyPage();
   /*
   Add action listener to cancel button of faxpropertypage object.
   */
   faxpropertypage.getCancelButton()addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
      if (oldcommname!=null)
      {
         faxpropertypage.setCommPort(oldcommname);
      }
      else if(oldcommname.length()>0)
      {
```

```
            faxpropertypage.setCommPort(oldcommname);
        }
        if (olddialing!=null)
        {
            faxpropertypage.setdialingMode(olddialing);
        }
        else if (olddialing.length()>0)
        {
            faxpropertypage.setdialingMode(olddialing);
        }
        if(oldmodemclass!=null)
        {
            faxpropertypage.setModemClass(oldmodemclass);
        }
        else if(oldmodemclass.length()>0)
        {
            faxpropertypage.setModemClass(oldmodemclass);
        }
        if(oldflowcontrol!=null)
        {
            faxpropertypage.setFlowControl(oldflowcontrol);
        }
        else if(oldflowcontrol.length()>0)
        {
            faxpropertypage.setFlowControl(oldflowcontrol);
        }
        faxpropertypage.setFlowControlIn(oldflowcontrolin);
    }
    });
    /*Initialize object of JFileChooser.*/
    filechooser=new JFileChooser();
    /*
    Declare and initialize object of GridBagLayout class
    */
    GridBagLayout gridbaglayout=new GridBagLayout();
    /*
    Declare and initialize object of GridBagConstraints class
    */
    GridBagConstraints gridbagconstraints =new GridBagConstraints();
    /*
    Initialize object of JPanel and set its layout as GridBagLayout.
    */
    toppanel=new JPanel(gridbaglayout);
    /*
    Initialize object of JLabel and set constraints to this label
    */
    modemstringLabel=new JLabel("Modem Initial String");
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(10, 10, 5, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemstringLabel, gridbagconstraints);
    /*Add label to panel*/
    toppanel.add(modemstringLabel);
    /*
    Initialize object of JTextField and set constraints to this label
    */
    modemstringtext=new JTextField("ATV1Q0");
    modemstringtext.setPreferredSize(new Dimension(150, 23));
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(10, 10, 5, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemstringtext, gridbagconstraints);
    /*Add button to panel*/
    toppanel.add(modemstringtext);
    /*
    Initialize object of JLabel and set constraints to this label
    */
    fexLabel=new JLabel("Fax Number");
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=1;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(0, 10, 10, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(fexLabel, gridbagconstraints);
    /*Add label to panel*/
    toppanel.add(fexLabel);
    /*
    Initialize object of JTextField and set constraints to this label
    */
    faxtextfield=new JTextField("0, 51610236");
    faxtextfield.setPreferredSize(new Dimension(150, 23));
```

```
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=1;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(0, 10, 10, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(faxtextfield,gridbagconstraints);
    /*Add button to panel*/
    toppanel.add(faxtextfield);
    /*
    Initialize object of JCheckBox and set constraints to this label
    */
    htmlcheckbox=new JCheckBox(" HTML");
    htmlcheckbox.setBorderPaintedFlat(true);
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=2;
    gridbagconstraints.gridy=2;
    gridbagconstraints.weightx=0.0;
    gridbagconstraints.gridwidth=2;
    gridbagconstraints.insets=new Insets(0, 10, 0, 0);
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(htmlcheckbox, gridbagconstraints);
    /*Add checkbox to panel*/
    toppanel.add(htmlcheckbox);
    /*Add panel to main window*/
    contentpane.add(toppanel, BorderLayout.NORTH);
    /*Initialize object of editorpane.*/
    faxeditor=new JEditorPane();
    /*Initialize object of scrollpane.*/
    editorscrollpane=new JScrollPane(faxeditor);
    /*Add scrollbar to main window.*/
    contentpane.add(editorscrollpane,BorderLayout.CENTER);
    /*
    Declare and initialize object of JPanel class and set its layout as GridLayout.
    */
    JPanel bottomlabelpanel=new JPanel(new GridLayout(1, 1, 0, 0));
    bottomlabelpanel.add(processlabel=new JLabel(""));
    processlabel.setForeground(Color.blue);
    /*
     and initialize object of JPanel class and set its layout as GridLayout.
    */Declare
    JPanel bottombuttonpanel=new JPanel(new GridLayout(1, 5, 10, 10));
    bottombuttonpanel.add(new JLabel(""));
    /*Initialize object of JButton.*/
    sendbutton=new JButton(" Send Fax ");
    /*Set button size.*/
    sendbutton.setPreferredSize(new Dimension(20,23));
    sendbutton.setMnemonic('s');
    sendbutton.setActionCommand("send");
    sendbutton.addActionListener(this);
    bottombuttonpanel.add(sendbutton);
    cancelbutton=new JButton(" Cancel ");
    /*Set button size.*/
    cancelbutton.setPreferredSize(new Dimension(20,23));
    cancelbutton.setMnemonic('c');
    cancelbutton.setActionCommand("cancel");
    cancelbutton.addActionListener(this);
    bottombuttonpanel.add(cancelbutton);
    closebutton=new JButton(" Close ");
    /*Set button size.*/
    closebutton.setPreferredSize(new Dimension(20, 23));
    closebutton.setMnemonic('l');
    closebutton.setActionCommand("close");
    closebutton.addActionListener(this);
    bottombuttonpanel.add(closebutton);
    bottombuttonpanel.add(new JLabel(""));
    /*
    Declare and initialize object of JPanel class and set its layout as GridLayout.
    */
    JPanel bottompanel=new JPanel(new GridLayout(2, 1));
    bottompanel.add(bottomlabelpanel);
    bottompanel.add(bottombuttonpanel);
    /*Set panel size.*/
    bottompanel.setPreferredSize(new Dimension(550, 50));
    /*Add panel to main window.*/
    contentpane.add(bottompanel,BorderLayout.SOUTH);
}
/*
actionPerformed - This method is called when the user clicks the Send or Close button,
selects file menu items.
Parameters: ae - an ActionEvent object containing details of the event.
Return Value: NA
 */
public void actionPerformed(ActionEvent ae)
{
    String actioncommand=ae.getActionCommand();
    /*
    This is executed when user clicks the Open menu item.
```

```
*/
if ("open".equals(actioncommand))
{
   int returnVal =filechooser.showOpenDialog(PortFax.this);
   if (returnVal==JFileChooser.APPROVE_OPTION)
    {
      try
      {
         File file = filechooser.getSelectedFile();
         InputStream inputStream = new FileInputStream(file);
         BufferedReader br = new BufferedReader(new InputStreamReader(inputStream));
         String strTemp="";
         faxeditor.setText("");
         while ((strTemp = br.readLine())!=null)
         {
            faxeditor.setText(faxeditor.getText()+strTemp+'\n');
         }
         inputStream.close();
         }
         catch( FileNotFoundException e )
         {
            System.out.println( "File not found" );
         }
         catch( IOException e )
         {
            System.out.println( "I/O error" );
         }
       }
}
/*
This is executed when user clicks the View Property menu item.
*/
else if("view".equals(actioncommand))
{
   oldcommname=faxpropertypage.getCommPortName();
   olddialing=faxpropertypage.getDialing();
   oldmodemclass=(String)faxpropertypage.modemclasscombo.getSelectedItem();
   oldflowcontrol=faxpropertypage.getFlowcontrol();
   oldflowcontrolin=faxpropertypage.getFlowControlIn();
   faxpropertypage.setVisible(true);
}
/*
This is executed when user clicks the Exit menu item.
*/
else if("exit".equals(actioncommand))
{
   System.exit(0);
}
/*
This is executed when user clicks the Send button.
*/
else if ("send".equals(actioncommand))
{
   sendbutton.setEnabled(false);
   closebutton.setEnabled(true);
   if (htmlcheckbox.isSelected())
   {
      faxdoc= new ConvertIntoHTML();
      String[] pag=new String[1];
      pag[0]=faxeditor.getText();
      ((ConvertIntoHTML) faxdoc).text=pag;
      ((ConvertIntoHTML) faxdoc).pageimage=createImage(800, 550);
   }
   else
   {
      faxdoc= new ConvertIntoTextImage();
      ((ConvertIntoTextImage) faxdoc).text=faxeditor.getText();
      ((ConvertIntoTextImage) faxdoc).pageimage=createImage(800, 550);
      ((ConvertIntoTextImage) faxdoc).prepare();
   }
   /* Set modem configuration*/
   sendfax=new SendingFax();
   sendfax.bitorder=true;
   sendfax.log=true;
   sendfax.debug=d;
   sendfax.faxclass=1;
   sendfax.resolution=sendfax.RESOLUTION_NORMAL;
   sendfax.timeout=60;
   if (faxpropertypage.getmodemclass()=="Class 1") sendfax.faxclass=2;
   if (faxpropertypage.getDialing()=="Class 2") sendfax.faxclass=20;
   sendfax.dialingmode=faxpropertypage.getDialing();
   sendfax.flowcontrol=faxpropertypage.getFlowControlIn();
   if (faxpropertypage.getFlowcontrol()!="")
   {
      sendfax.flowcontrolin=(String)faxpropertypage.getFlowcontrol();
   }
   sendfax.setPortName(faxpropertypage.getCommPortName());
   sendfax.modemFBOR=true;
```

```
            if (modemstringtext.getText().length()>0) sendfax.setInitString(modemstringtext.getText());
            sendfax.listener=this;
            if (sendfax.openForFax(faxdoc))
            {
               if (sendfax.sendFax(faxtextfield.getText()))
               {
                  this.processlabel.setText("Fax successfully sent");
                  sendbutton.setEnabled(true);
                }
            }
         }
         /*
         This is executed when user clicks the Close button.
         */
         else if("cancel".equals(actioncommand))
         {
            if (sendfax!=null) sendfax.close();
            processlabel.setForeground(java.awt.Color.blue);
            processlabel.setText("Inactive...");
            sendbutton.setEnabled(true);
            closebutton.setEnabled(false);
            sendfax.serialPort.close();
         }
            /*
            This is executed when user clicks the Cancel button.
            */
            else if("close".equals(actioncommand))
            {
               System.exit(0);
            }
      }
      /*
      faxProgress - This method is called to show status of fax sending operation in the label.
      Parameters: status ,page
      Return Value:
       */
      public void faxProgress(int status,int page)
      {
         if (status==ST_CLOSE) processlabel.setText("Closing...");
         if (status==ST_CONNECTING) processlabel.setText("Connecting...");
         if (status==ST_CONVERT_FIlES) processlabel.setText("Converting fax files...");
         if (status==ST_INIT_MODEM) processlabel.setText("Initializing modem...");
         if (status==ST_OPEN_PORT) processlabel.setText("Opening port...");
         if (status==ST_SEND_PAGE) processlabel.setText("Sending page " + (int) (page +1));
         if (status==FaxStatusListener.ST_REC_PAGE) processlabel.setText("Receiving page " + (int) (pag
         if (status==FaxStatusListener.ST_REC_CALL) processlabel.setText("Call detected...");
         if (status==FaxStatusListener.ST_WAIT_CALL) processlabel.setText("Waiting for call...");
         this.paintAll(this.getGraphics());
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortFax class. This class generates the main window of the Fax application, as shown in Figure 6-2:

Figure 6-2: The Main Window of the Fax Application

The File menu provides three menu items: Open, View Properties, and Exit. The Open command opens the file selected by the end user in the edit pane of the Fax application. The View Properties command displays a user interface to set the properties of the COM port. The Fax application uses the FaxPropertyPage.jave file to create the user interface. The Exit command terminates the Fax application.

The Modem Initial String text box allows an end user to set the initial command string for the modem. The Fax Number text box specifies the number to which the end user wants to send the fax. The HTML check box enables an end user to send the fax as an HTML page. If the HTML check box is not selected, the Fax application sends the Fax as a text image.

The various methods defined in the PortFax.java File are:

- createGUI():Creates the user interface of the Fax application.
- actionPerformed():Acts as an event listener and activates an appropriate class or method based on the button that the end user clicks. For example, when the end user selects the HTML check box and clicks the Send Fax button, the actionPerformed() method creates an object of the ConvertIntoHTML class, which converts the text file into an image of an HTML page. If the end user does not select the HTML check box and clicks the Send Fax button, the actionPerformed() method creates an object of the ConvertIntoTextImage class. This class converts the file that is open in the edit pane to a text image. When the end user clicks the Close button of the Fax application, the actionPerformed() method closes the main window.
- faxProgress():Displays the status of the Fax application.

◄ PREVIOUS NEXT ►

Converting the File into an Image

The ConvertIntoHTML.java and ConvertIntoTextImage.java files convert the file that is open in the user interface of the Fax application to images of HTML and text pages, respectively. The ConvertIntoHTML.java and ConvertIntoTextImage.java files implement the interface defined by the PreparedFaxDoc.java file.

Listing 6-2 shows the PreparedFaxDoc.java file:

Listing 6-2: The PreparedFaxDoc.java File

```
/*Imports required java.awt.Image class.*/
import java.awt.Image;
/*
interface PreparedFaxDoc - This interface declare an abstract method.
Methods:
   getFaxPage
*/
public interface PreparedFaxDoc
{
    public abstract Image getFaxPage(int i);
}
```

Download this listing.

In the above listing, the getFaxPage()method returns the converted image. The image returned by the getFaxPage()method shows the contents of the file sent as a fax.

The ConvertIntoTextImage.java file converts the file opened in the edit pane to a text image. This image contains the contents of the file.

Listing 6-3 shows the ConvertIntoTextImage.java file:

Listing 6-3: The ConvertIntoTextImage.java File

```
/*Imports required javax.comm package classes.*/
import java.awt.*;
/*
class ConvertIntoTextImage - This class is used to convert editpane Contents into text page
Constructor:
   ConvertIntoTextImage - This constructor initializes parameters
Methods:
   getFaxPage - Returns fax Page.
   getTextImage - Returns text Page.
getTextString - This method is called to convert String into String array.
prepare - This method is called to invoke getTextString method.
*/
public class ConvertIntoTextImage implements PreparedFaxDoc
{
public Font textfont;
       public String text;
       public int linespage;
       public int topmargin;
       public int leftmargin;
       public Image pageimage;
       private String textstring[];
       private int linenumber;
       public ConvertIntoTextImage()
       {
           textfont = new Font("Serif", 0, 12);
           text = "";
           linespage = 66;
           topmargin = 4;
           leftmargin = 4;
           pageimage = null;
           textstring = new String[1000];
           linenumber = 0;
    }
   /*
   getFaxPage - This method is called to return fax Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    public Image getFaxPage(int page)
    {
        if(page * linespage >= linenumber)
            return null;
        else
            return getTextImage(page);
    }
   /*
   getTextImage - This method is called to return text Page.
   Parameters: page - Variable of integer type.
```

```
   Return Value: Image
   */
    private Image getTextImage(int page)
    {
         Graphics g = pageimage.getGraphics();
         g.setColor(Color.white);
         g.fillRect(0, 0, pageimage.getWidth(null), pageimage.getHeight(null));
         g.setColor(Color.black);
         g.setFont(textfont);
         int lineH = g.getFontMetrics().getHeight();
         int charW = g.getFontMetrics().stringWidth("X");
         int linesImage = pageimage.getHeight(null) / lineH - topmargin;
         if(linesImage < linespage)
         linespage = linesImage;
         int currentY = lineH * topmargin;
         int currentLine = 0;
         for(int currentTotalLine = page * linespage; currentLine < linespage &&
         currentTotalLine < linenumber; currentTotalLine++)
         {
            if(textstring[currentTotalLine] != null)
            g.drawString(textstring[currentTotalLine], leftmargin * charW, currentY);
            currentY += lineH;
            currentLine++;
         }
         return pageimage;
       }
      /*
      prepare - This method is called to invoke getTextString method.
      Parameters:  NA
      Return Value: NA
      */
      public void prepare()
       {
        getTextString(text);
       }
      /*
      getTextString - This method is called to convert String into String array.
      Parameters: text -Object of String object.
      Return Value: NA
      */
       private void getTextString(String text)
       {
          int p = 0;
          linenumber = 0;
          for(p = text.indexOf("\n"); p >= 0; p = text.indexOf("\n"))
          {
            textstring[linenumber++] = text.substring(0, p);
            text = text.substring(p + 1, text.length());
          }
        textstring[linenumber++] = text;
    }
}
```

Download this listing.

In the above listing, the ConvertIntoTextImage class converts the text of the file into a image. The methods defined in the above listing are:

- `getFaxPage()`:Returns the page as an image to send as a fax. This method is an abstract method that is defined in the PreparedFaxDoc.java interface.
- `getTextImage()`:Creates an object of the Graphics class. The method converts the file opened in the edit pane to a text image.
- `prepare()`:Invokes the getTextString() method.
- `getTextString()`:Retrieves the contents of the text file that is open in the edit pane into a String array.

The ConvertIntoHTML.java file converts the file opened in the edit pane to an image of the HTML page sent as a fax.

Listing 6-4 shows the ConvertIntoHTML.java file:

Listing 6-4: The ConvertIntoHTML.java File

```
/*Imports required java.awt package classes*/
import java.awt.*;
/*Imports required javax.swing classes*/
import javax.swing.JComponent;
import javax.swing.JEditorPane;
import javax.swing.text.JTextComponent;
/*
class ConvertIntoHTML - This class is the use to convert editpane Contents into HTML image.
Constructor:
ConvertIntoHTML - This constructor initializes parameters
Methods:
   getFaxPage - Returns fax Page
   getHTMLPage - Returns HTML Page
```

```
*/
public class ConvertIntoHTML implements PreparedFaxDoc
{
    public String text[];
    public Image pageimage;
    private JEditorPane htmleditorpanel;
    public ConvertIntoHTML()
    {
        pageimage = null;
        htmleditorpanel = new JEditorPane();
    }
   /*
   getFaxPage - This method is called to return fax Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    public Image getFaxPage(int page)
    {
       if(text == null)
            return null;
        if(page >= text.length)
            return null;
        else
            return getHTMLPage(page);
    }
   /*
   getHTMLPage - This method is called to return HTML Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    private Image getHTMLPage(int page)
    {
         Graphics g = pageimage.getGraphics();
         htmleditorpanel.setSize(pageimage.getWidth(null), pageimage.getHeight(null));
         htmleditorpanel.setBackground(Color.white);
         htmleditorpanel.setEditable(false);
         htmleditorpanel.setContentType("text/html");
         htmleditorpanel.setText(text[page]);
         htmleditorpanel.paint(g);
         return pageimage;
    }
}
```

Download this listing.

In the above listing, the ConvertIntoHTML.java file implements the PreparedFaxDoc.java interface. The various methods defined in the ConvertIntoHTML.java file are:

- `getFaxPage()`:Returns the page as an HTML page to send as a fax. This method is an abstract method that is defined in the PreparedFaxDoc.java interface.
- `getHTMLPage()`:Converts the HTML page of the text file open in the user interface of the Fax application to an image.

PREVIOUS NEXT

Displaying Port Properties

The FaxPropertyPage.java file displays the properties of the port to which the modem is connected. The modem helps send the fax to the fax number of the destination. The FaxPropertyPage.java file creates a user interface to display and set the properties of the port.

Listing 6-5 shows the FaxPropertyPage.java file:

Listing 6-5: The FaxPropertyPage.java File

```
/*Imports required Comm classes*/
import javax.comm.*;
/*Imports required javax.swing package classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required java.util package classes*/
import java.util.*;
/*
class FaxPropertyPage - This class is the Port Property dialog window,
Enable the end user to change the property of port.
Constructor:
   FaxPropertyPage - This constructor creates GUI.
Methods:
   setParity - Sets Parity bit.
   setStopbits - Sets Stop bit.
   setDatabits - Set Data bit
   setBaudRate - Sets Baud rate.
getCancelButton - Returns object of cancel button.
   getOkButton -  Returns object of ok button.
getFlowControlOut - Returns flow control out value.
getFlowControlIn - Returns flow control in value.
   getParity - Returns Parity bit value.
   getDatabits - Returns Data bit value.
   getStopbits - Returns Stop bit value.
   getBaudRate - Returns Baud rate value.
   getCommPortName - Returns  port name.
   getCommPort -  Returns object of CommPort class
*/
class FaxPropertyPage extends JFrame implements ActionListener
{
   /*Declare object of Enumeration class.*/
   Enumeration listport;
   /*Declare object of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /*Declare object of JPanel class.*/
   JPanel buttonpanel;
   JPanel propertylist;
   /*Declare object of JLabel class.*/
   JLabel portlistlabel;
   JLabel flowlabel;
   JLabel paritylabel;
   JLabel dialinglabel;
   JLabel modemclasslabel;
   JLabel flowcontrollabel;
   JLabel stoplabel;
   /*Declare object of JComboBox class.*/
   JComboBox comportcombo;
   JComboBox flowcombo;
   JComboBox modemclasscombo;
   JComboBox dialingcombo;
   JComboBox flowcontrolcombo;
   /*
   Declare object of CommPortIdentifier class.
   */
   CommPortIdentifier portId;
   public FaxPropertyPage()
   {
      /* Set title of main window. */
      super("Fax Property Page");
      /*
      Declare and initialize object of Container class.
      */
      Container contentpane=getContentPane();
      comportcombo=new JComboBox();
      listport=CommPortIdentifier.getPortIdentifiers();
      while (listport.hasMoreElements())
      {
         portId=(CommPortIdentifier)listport.nextElement();
         if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
```

```
            comportcombo.addItem(portId.getName());
        }
    }
    /*Initialize Object of JComboBox. */
    flowcombo=new JComboBox();
    flowcombo.addActionListener(this);
    /*
    Declare and initialize object of GridBagLayout class
    */
    GridBagLayout gridbaglayout=new GridBagLayout();
    /*
    Declare and initialize object of GridBagConstraints class
    */
    GridBagConstraints gridbagconstraint=new GridBagConstraints();
    /*
    Initialize object of JPanel class and set its layout as gridbaglayout.
    */
    propertylist=new JPanel(gridbaglayout);
    /*Set background color as white.*/
    propertylist.setBackground(Color.white);
    /*
    Initialize object of JLabel class and gives its name.
    */
    Portlistlabel=new JLabel("Port Name",JLabel.RIGHT);
    /*
    Initialize object of JLabel class and gives its name.
    */
    flowlabel=new JLabel("Flow Control In");
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbagconstraint.insets=new Insets(5,5,5,5);
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=0;
    gridbagconstraint.weightx=1.0;
    gridbagconstraint.weighty=1.0;
    gridbaglayout.setConstraints(portlistlabel,gridbagconstraint);
    /* Add portlistlabel lable to panel. */
    propertylist.add(portlistlabel);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
    gridbagconstraint.gridy=0;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbagconstraint.insets=new Insets(5,5,5,5);
    gridbaglayout.setConstraints(comportcombo,gridbagconstraint);
    /*Add comportcombo combo box to panel.*/
    propertylist.add(comportcombo);
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=1;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(flowlabel,gridbagconstraint);
    /*Add flowlabel lable to panel.*/
    propertylist.add(flowlabel);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
    gridbagconstraint.gridy=1;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(flowcombo,gridbagconstraint);
    /*Add comportcombo combo box to panel.*/
    propertylist.add(flowcombo);
    /*
    Initialize object of JLabel class and gives its name.
    */
    modemclasslabel=new JLabel("Modem Class",JLabel.RIGHT);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=2;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemclasslabel,gridbagconstraint);
    /*Add modemclasslabel label to panel.*/
    propertylist.add(modemclasslabel);
    /*
    Initialize object of JComboBox class.
    */
    modemclasscombo=new JComboBox();
    modemclasscombo.addActionListener(this);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
```

```
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(modemclasscombo,gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(modemclasscombo);
/*
Initialize object of JLabel class and gives its name.
*/
dialinglabel=new JLabel("Dialing",JLabel.RIGHT);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(dialinglabel,gridbagconstraint);
/*Add dialinglabel label to panel.*/
propertylist.add(dialinglabel);
/*
Initialize object of JComboBox class.
*/
dialingcombo=new JComboBox();
dialingcombo.addActionListener(this);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(dialingcombo,gridbagconstraint);
/*Add dialingcombo combo to panel.*/
propertylist.add(dialingcombo);
/*
Initialize object of JLabel class and gives its name.
*/
flowcontrollabel=new JLabel("Flow Control Command",JLabel.RIGHT);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrollabel,gridbagconstraint);
/*
Add flowcontrollabel label to panel.
*/
propertylist.add(flowcontrollabel);
/*
Initialize object of JComboBox class.
*/
flowcontrolcombo=new JComboBox();
flowcontrolcombo.addActionListener(this);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrolcombo,gridbagconstraint);
/*
Add flowcontrolcombo combo to panel.
*/
propertylist.add(flowcontrolcombo);
contentpane.add(propertylist);
/*Fill dialingcombo combo box.*/
dialingcombo.addItem("Tone");
dialingcombo.addItem("Pulse");
dialingcombo.setSelectedItem("Tone");
/*Fill flowcombo combo box.*/
flowcombo.addItem("none");
flowcombo.addItem("RtsCts");
flowcombo.addItem("Xon/Xoff In");
flowcombo.setSelectedItem("none");
/*Fill flowcontrolcombo combo box.*/
flowcontrolcombo.addItem("");
flowcontrolcombo.addItem("-- RtsCts --");
flowcontrolcombo.addItem("-- XonXoff --");
flowcontrolcombo.addItem("AT+FLO=2");
flowcontrolcombo.addItem("AT&K3");
flowcontrolcombo.addItem("AT&\\Q3");
flowcontrolcombo.addItem("AT+FLO=1");
flowcontrolcombo.addItem("AT&K4");
flowcontrolcombo.addItem("AT&\\Q4");
flowcontrolcombo.setSelectedItem("");
/*Fill modemclasscombo combo box.*/
modemclasscombo.addItem("Class 1");
modemclasscombo.addItem("Class 2");
```

```
   modemclasscombo.addItem("Class 2.0");
   modemclasscombo.setSelectedItem("Class 1");
   /*
   Initialize object of JButton and set its label.
   */
   okbutton=new JButton("OK");
   okbutton.setActionCommand("ok");
   okbutton.addActionListener(new ActionListener()
   {public void actionPerformed(ActionEvent ae){setVisible(false);}});
   /*
   Initialize object of JButton and set its label
   */
   cancelbutton=new JButton("Cancel");
   okbutton.addActionListener(new ActionListener()
   {public void actionPerformed(ActionEvent ae){setVisible(false);}});
   cancelbutton.setActionCommand("cancel");
   cancelbutton.addActionListener(this);
   /* Initialize object of JPanel. */
   buttonpanel=new JPanel();
   /*
   Set buttonpanel  layout as gridbaglayout.
   */
   buttonpanel.setLayout(gridbaglayout);
   /*
   Set constraints for gridbagconstraint.
   */
   gridbagconstraint.fill = GridBagConstraints.HORIZONTAL;
   gridbagconstraint.insets=new Insets(5,5,5,5);
   gridbagconstraint.gridx=0;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(okbutton,gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(okbutton);
   /*
   Set constraints for gridbagconstraint.
   */
   gridbagconstraint.gridx=1;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(cancelbutton,gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(cancelbutton);
   contentpane.add(buttonpanel,BorderLayout.SOUTH);
   addWindowListener(new WindowAdapter()
   {
      public void windowClosed(WindowEvent we)
      {
         setVisible(false);
      }
   });
   /*Set frame size.*/
   setSize(400,200);
}
/*
actionPerformed - This methods is executed when the end user selects items from combo box or clic
Parameters: e - Object of ActionEvent class
Return Value: NA
*/
public void actionPerformed(ActionEvent e)
{
   if (e.getSource() instanceof JButton)
   {
      setVisible(false);
   }
   else
   {
      JComboBox cb = (JComboBox)e.getSource();
      String petName = (String)cb.getSelectedItem();
   }
}
/*
getCommPort - Gives object of  CommPort class
Parameters: NA
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getCommPort()
{
   CommPortIdentifier comp=null;
   try
   {
      comp= javax.comm.CommPortIdentifier.getPortIdentifier((String)comportcombo.getSelectedItem(
   }
   catch(NoSuchPortException e)
   {
      JOptionPane.showMessageDialog(null,"open","Port not exists",JOptionPane.PLAIN_MESSAGE);
   }
```

```
   return comp;
}
/*
getCommPortName - Gives name of selected port.
Parameters: NA
Return Value: String
*/
public String getCommPortName()
{
   return (String)comportcombo.getSelectedItem();
}
/*
getDialing - Gives dialing mode.
Parameters: NA
Return Value: String
*/
public String getDialing()
{
   return (String)dialingcombo.getSelectedItem();
}
/*
getmodemclass - Gives modem class.
Parameters: NA
Return Value: String
*/
public String getmodemclass()
{
   return (String)modemclasscombo.getSelectedItem();
}
/*
getFlowControlIn - Gives type of flow in control.
Parameters: NA
Return Value: int
*/
public int getFlowControlIn()
{
   String flowcontrol=(String)flowcombo.getSelectedItem();
   if (flowcontrol.equals("none"))
   {
      return 0;
   }
   if (flowcontrol.equals("RtsCts"))
   {
      return 2;
   }
   if (flowcontrol.equals("Xon/Xoff In"))
   {
      return 1;
   }
   return 0;
}
/*
getCancelButton - Gives object of JButton class.
Parameters: NA
Return Value: JButton
*/
public JButton getCancelButton()
{
   return cancelbutton;
}

public void setCommPort(String commname)
{
   comportcombo.setSelectedItem(commname);
}
public void setdialingMode(String dialing)
{
   dialingcombo.setSelectedItem(dialing);
}
public void setModemClass(String modemclass)
{
   modemclasscombo.setSelectedItem(modemclass);
}
public void setFlowControl(String flowcontrol)
{
   flowcontrolcombo.setSelectedItem(flowcontrol);
}
public void setFlowControlIn(int flowcontrolin)
{
   if(flowcontrolin==0)
   {
      flowcombo.setSelectedItem("none");
   }
   else if (flowcontrolin==1)
   {
      flowcombo.setSelectedItem("RtsCts")
   }
```

```
            else
            {
               flowcombo.setSelectedItem("Xon/Xoff In");
            }
        }
        /*
        getFlowcontrol - Gives flow control.
        Parameters: NA
        Return Value: String
        */
        public String getFlowcontrol()
        {
            return (String)flowcontrolcombo.getSelectedItem();
        }
}
```

Download this listing.

In the above listing, the constructor of the FaxPropertyPage class creates the user interface of the FaxPropertyPage.java file, as shown in Figure 6-3:

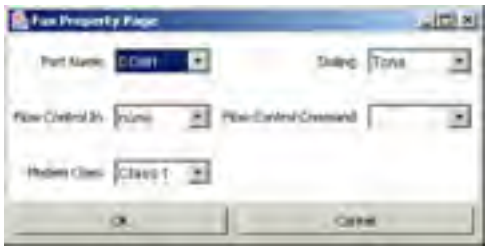

Figure 6-3: The Fax Property Page Window

The various methods defined in the FaxPropertyPage.java file are:

- `actionPerformed()`:Acts as an event listener and activates an appropriate class or method based on the button that an end user clicks. When the end user clicks the OK button, the actionPerformed() method sets the properties of the port to the values selected on the Fax Property Page window that the FaxPropertyPage class creates. When an end user clicks the Cancel button, the actionPerformed() method closes the Fax Property Page window.
- `getCommPort()`: Determines whether the selected COM port exists or not.
- `setCommPort()`:Sets the value of the selected COM port.
- `getDialing()`:Returns the dialing mode of the selected COM port, such as?.
- `setdialingMode()`:Sets the dialing mode property for the selected COM port.
- `getmodemclass()`:Returns the class of the modem that helps send the fax.
- `setModemClass()`:Sets the value of the modem class to the selected value.
- `getFlowcontrol()`:Returns the value of the Flow control property of the selected COM port. The Flow control property indicates data flow from the modem to the selected COM port.
- `setFlowControl()`:Sets the value of the Flow Control property of the selected COM port.
- `getFlowControlIn()`:Returns the value of the Flow Control In property of the selected COM port. The Flow Control In property indicates data flow from the selected COM port to the modem.
- `setFlowControlIn`:Sets the value of the Flow Control In property of the selected COM port.
- `getCommPortName()` Returns the name of the COM port that sends the fax.
- `getCancelButton()`:Creates and returns an instance of the Cancel button.

Displaying the Status of the Fax Application

The FaxStatusListener.java file sets integer values for the status of the Fax application.

Listing 6-6 shows the FaxStatusListener.java file:

Listing 6-6: The FaxStatusListener.java File

```
public interface FaxStatusListener
{
    public abstract void faxProgress(int i, int j);
    public static final int ST_OPEN_PORT = 1;
    public static final int ST_INIT_MODEM = 2;
    public static final int ST_CONNECTING = 3;
    public static final int ST_SEND_PAGE = 4;
    public static final int ST_CLOSE = 5;
    public static final int ST_CONVERT_FIlES = 6;
    public static final int ST_REC_PAGE = 4;
    public static final int ST_REC_CALL = 7;
    public static final int ST_WAIT_CALL = 8;
}
```

Download this listing.

In the above listing, the FaxStatusListener.java file declares an abstract method, faxProgress(). This method accepts two integer parameters. The first parameter indicates the status of the fax and the second parameter indicates the page to send as a fax. The PortFax.java file implements the FaxStatusListener interface to display the current status of the Fax application.

Setting Modem Capabilities

The ModemCapabilities.java file sets the encoding and decoding capabilities of a modem.

Listing 6-7 shows the ModemCapabilities.java file:

Listing 6-7: The ModemCapabilities.java File

```
/*Imports required java.util.Vector class*/
import java.util.Vector;
/*
class ModemCapabilities - This class is the main class of the application.
This class initializes the interface and loads all components like the menubar,
textareas, and buttons before displaying the result.
Constructor:
ModemCapabilities - This constructor initializes various modem parameters
Methods:
encodeCapabilities - This method is called to ecode modem capabilities
decodeCapabilitiesClass2 - Find modem capabilities of class2 type.
decodeCapabilities -  Find modem capabilities of class1 type.
*/
public class ModemCapabilities
{
public String vRate;
public String recRate;
   public boolean t4;
       public int resolution;
       public int rate;
       public int length;
       public int width;
       public int huffman;
       public int scanTime;
       public int errorCorrection;
       public boolean binaryTransfer;
       private static int itemlength = 0;
       private static int itemsize = 1;
       private static int seconditemsize = 2;
       private static int thirditemsize = 3;
       private static int forthitemsize = 4;
       private static int fifthitemsize = 5;
       private static int sixthitemsize = 6;
       private static int seventhitemsize = 7;
       public ModemCapabilities()
       {
           t4 = true;
           resolution = 0;
           rate = 96;
           vRate = "V29";
           recRate = "";
           length = 0;
           width = 1728;
           huffman = 1;
           scanTime = 0;
           errorCorrection = 0;
           binaryTransfer = false;
       }
      /*
      encodeCapabilities - This method is called to ecode modem capabilities
      Parameters: NA
      Return Value: byte[]
      */
       public byte[] encodeCapabilities()
       {
            byte bi[] = new byte[3];
          bi[0] = 0;
          bi[1] = 0;
          bi[2] = 0;
           bi[1] = 64;
                byte tmp = 0;
           if(rate == 145 && vRate.compareTo("V17") == 0)
                tmp = 4;
           if(rate == 145 && vRate.compareTo("V33") == 0)
                tmp = 8;
           if(rate == 121 && vRate.compareTo("V17") == 0)
                tmp = 20;
           if(rate == 121 && vRate.compareTo("V33") == 0)
                tmp = 24;
           if(rate == 96 && vRate.compareTo("V17") == 0)
                tmp = 36;
           if(rate == 72 && vRate.compareTo("V17") == 0)
                 tmp = 52;
           if(rate == 72 && vRate.compareTo("V33") == 0)
                 tmp = 48;
           if(rate == 72 && vRate.compareTo("V29") == 0)
```

```
                tmp = 48;
          if(rate == 96 && vRate.compareTo("V29") == 0)
          tmp = 32;
        if(rate == 48)
          tmp = 16;
        if(rate == 24)
          tmp = 0;
        tmp &= 0xff;
        bi[1] = (byte)(bi[1] | tmp);
        if(resolution == 1)
           bi[1] = (byte)(bi[1] | 2);
        if(huffman == 2)
             bi[1] = (byte)(bi[1] | 1);
         tmp = 0;
         if(width == 1728)
             tmp = 0;
         if(width == 2432)
             tmp = 64;
         if(width == 2048)
             tmp = -128;
         tmp &= 0xff;
         bi[2] = (byte)(bi[2] | tmp);
         tmp = 0;
         if(length == 2)
             tmp = 16;
         if(length == 1)
             tmp = 32;
         tmp &= 0xff;
        bi[2] = (byte)(bi[2] | tmp);
         tmp = 0;
         if(scanTime == 20)
             tmp = 0;
         if(scanTime == 40)
             tmp = 2;
         if(scanTime == 10)
             tmp = 4;
         if(scanTime == 5)
             tmp = 8;
         if(scanTime == 0)
             tmp = 14;
         tmp &= 0xff;
         bi[2] = (byte)(bi[2] | tmp);
         return bi;
 }
/*
decodeCapabilitiesClass2 - This method is called to decode capabilities of class2 type mode
Parameters: ae - an ActionEvent object containing details of the event.
Return Value:boolean
 */
public boolean decodeCapabilitiesClass2(String str)
{
   Vector items = new Vector();
   String item = "";
   for(int p = str.indexOf(","); p >= 0; p = str.indexOf(","))
   {
         item = "";
         if(p > 0)
             item = str.substring(0, p);
         str = str.substring(p + 1, str.length());
         items.addElement(item);
     }
     if(items.size() > itemlength)
     {
         item = (String)items.elementAt(itemlength);
          if(item.trim().equals("0"))
          resolution = 0;
          if(item.trim().equals("1"))
          resolution = 1;
    }
     if(items.size() > itemsize)
    {
         item = (String)items.elementAt(itemsize);
         if(item.trim().equals("0"))
         {
             rate = 24;
             vRate = "V27";
         }
         if(item.trim().equals("1"))
         {
             rate = 48;
             vRate = "V27";
         }
         if(item.trim().equals("2"))
         {
             rate = 72;
             vRate = "V29";
         }
         if(item.trim().equals("3"))
```

```
                    {
                        rate = 96;
                        vRate = "V29";
                    }
                    if(item.trim().equals("4"))
                    {
                        rate = 121;
                        vRate = "V17";
                    }
                    if(item.trim().equals("5"))
                    {
                        rate = 145;
                        vRate = "V17";
                    }
                }
                if(items.size() > seconditemsize)
                {
                    item = (String)items.elementAt(seconditemsize);
                    width = 1728;
                    if(item.trim().equals("0"))
                    width = 1728;
                    if(item.trim().equals("1"))
                    width = 2048;
                    if(item.trim().equals("2"))
                    width = 2432;
            }
            if(items.size() > thirditemsize)
            {
                item = (String)items.elementAt(thirditemsize);
                if(item.trim().equals("0"))
                length = 0;
                if(item.trim().equals("1"))
                length = 1;
                if(item.trim().equals("2"))
                length = 2;
            }
            if(items.size() > forthitemsize)
            {
                item = (String)items.elementAt(forthitemsize);
                huffman = 2;
                if(item.trim().equals("0"))
                huffman = 1;
            }
            if(items.size() > fifthitemsize)
            {
                item = (String)items.elementAt(fifthitemsize);
                errorCorrection = 0;
                if(item.trim().equals("1"))
                errorCorrection = 1;
                if(item.trim().equals("2"))
                errorCorrection = 2;
            }
            if(items.size() > sixthitemsize)
            {
                item = (String)items.elementAt(sixthitemsize);
                binaryTransfer = false;
                if(item.trim().equals("1"))
                binaryTransfer = true;
            }
            if(items.size() > seventhitemsize)
            {
                item = (String)items.elementAt(seventhitemsize);
                if(resolution == 0)
                {
                    if(item.trim().equals("0"))
                        scanTime = 0;
                    if(item.trim().equals("1"))
                        scanTime = 5;
                    if(item.trim().equals("2"))
                        scanTime = 10;
                    if(item.trim().equals("3"))
                        scanTime = 20;
                    if(item.trim().equals("4"))
                        scanTime = 20;
                    if(item.trim().equals("5"))
                        scanTime = 20;
                    if(item.trim().equals("6"))
                        scanTime = 40;
                    if(item.trim().equals("7"))
                        scanTime = 40;
                }
else
                {
                    if(item.trim().equals("0"))
                        scanTime = 0;
                    if(item.trim().equals("1"))
                        scanTime = 5;
                    if(item.trim().equals("2"))
```

```
                scanTime = 10;
            if(item.trim().equals("3"))
                scanTime = 10;
            if(item.trim().equals("4"))
                scanTime = 10;
            if(item.trim().equals("5"))
                scanTime = 20;
            if(item.trim().equals("6"))
                scanTime = 20;
            if(item.trim().equals("7"))
                scanTime = 40;
        }
    }
    return true;
 }
/*
decodeCapabilities - This method is called to decode capabilities of class1 type mode
Parameters:   ae - an ActionEvent object containing details of the event.
Return Value:boolean
 */
 public boolean decodeCapabilities(byte bi[])
 {
    int b = bi[1];
    b &= 0x40;
    if(b == 64)
        t4 = true;
    else
        t4 = false;
    b = bi[1];
    b &= 0x3c;
    rate = 96;
    vRate = "V29";
    if(b == 56)
    {
        rate = 96;
        vRate = "V33";
    }
    if(b == 52)
    {
        rate = 145;
        vRate = "V17";
    }
    if(b == 48)
    {
        rate = 72;
        vRate = "V29";
    }
    if(b == 32)
    {
        rate = 96;
        vRate = "V29";
    }
    if(b == 16)
    {
        rate = 48;
        vRate = "V17";
    }
    if(b == 0)
    {
        rate = 24;
        vRate = "V27FB";
    }
    b = bi[1];
    b &= 2;
    if(b == 2)
        resolution = 1;
    b = bi[1];
    b &= 1;
    if(b == 1)
        huffman = 2;
    b = bi[2];
    b &= 0xc0;
    if(b == 0)
        width = 1728;
    if(b == 64)
        width = 2432;
    if(b == 128)
        width = 2048;
    if(b == 192)
        width = 2432;
    b = bi[2];
    b &= 0x30;
    if(b == 0)
        length = 0;
    if(b == 16)
        length = 2;
    if(b == 32)
        length = 1;
```

```
            b = bi[2];
            b &= 0xe;
            if(b == 0)
                scanTime = 20;
            if(b == 2)
                scanTime = 40;
            if(b == 4)
                scanTime = 10;
            if(b == 6)
                scanTime = 10;
            if(b == 8)
                scanTime = 5;
            if(b == 14)
                scanTime = 0;
            if(b == 10)
                scanTime = 40;
            if(b == 12)
                scanTime = 20;
            return true;
        }
}
```

Download this listing.

In the above listing, the ModemCapabilities class sets the encoding and decoding capabilities of a modem, such as time to scan the document and transfer rate to send the fax. The various methods defined in the Capabilities class are:

- `encodeCapabilities()`:Sets the encoding capabilities of a modem.
- `decodeCapabilitiesClass2()`:Sets the decoding capabilities of a class 2 modem.
- `decodeCapabilities()`:Sets the decoding capabilities of a modem.

PREVIOUS NEXT

Encoding the Fax Document

The ImageToFaxEncoder.java encodes the fax document that is currently opened in the edit pane of the Fax Application window.

Listing 6-8 shows the ImageToFaxEncoder.java file:

Listing 6-8: The ImageToFaxEncoder.java File

```
import java.awt.Image;
import java.awt.image.PixelGrabber;
import java.io.PrintStream;
public class ImageToFaxEncoder
{
public boolean debug;
       public int minBytesLine;
       public boolean alignEOL;
       public boolean completeLine;
       public int lineWidth;
       public int pageLength;
       public boolean completePage;
       public boolean scaleImage;
       public int scaleFactor;
       public boolean centerImage;
       public int whiteColor;
       public int blackColor;
       private int EOLimagearray[] = {
         86, 101, 86, 102, 86, 103, 86, 104, 86, 105,86, 106, 86, 107, 86, 108, 86, 114, 86, 115,
         86, 116, 86, 117, 86, 118, 86, 119, 86, 120,86, 121, 86, 122, 86, 123, 86, 124, 86, 130,
         86, 131, 86, 132, 86, 138, 86, 139, 86, 150,86, 151, 87, 101, 87, 102, 87, 103, 87, 104,
         87, 105, 87, 106, 87, 107, 87, 108, 87, 109,87, 110, 87, 114, 87, 115, 87, 116, 87, 117,
         87, 118, 87, 119, 87, 120, 87, 121, 87, 122,87, 123, 87, 124, 87, 130, 87, 131, 87, 132,
         87, 138, 87, 139, 87, 140, 87, 149, 87, 150,87, 151, 88, 101, 88, 102, 88, 109, 88, 110,
         88, 114, 88, 115, 88, 129, 88, 130, 88, 132,88, 133, 88, 139, 88, 140, 88, 141, 88, 148,
         88, 149, 88, 150, 89, 101, 89, 102, 89, 110,89, 111, 89, 114, 89, 115, 89, 129, 89, 130,
         89, 132, 89, 133, 89, 140, 89, 141, 89, 142,89, 147, 89, 148, 89, 149, 90, 101, 90, 102,
         90, 110, 90, 111, 90, 114, 90, 115, 90, 128,90, 129, 90, 133, 90, 134, 90, 141, 90, 142,
         90, 143, 90, 146, 90, 147, 90, 148, 91, 101,91, 102, 91, 110, 91, 111, 91, 114, 91, 115,
         91, 128, 91, 129, 91, 133, 91, 134, 91, 142,91, 143, 91, 144, 91, 145, 91, 146, 91, 147,
         92, 101, 92, 102, 92, 109, 92, 110, 92, 114,92, 115, 92, 116, 92, 117, 92, 118, 92, 119,
         92, 120, 92, 121, 92, 122, 92, 123, 92, 128,92, 129, 92, 133, 92, 134, 92, 143, 92, 144,
         92, 145, 92, 146, 93, 101, 93, 102, 93, 103,93, 104, 93, 105, 93, 106, 93, 107, 93, 108,
         93, 109, 93, 110, 93, 114, 93, 115, 93, 116,93, 117, 93, 118, 93, 119, 93, 120, 93, 121,
         93, 122, 93, 123, 93, 127, 93, 128, 93, 134,93, 135, 93, 143, 93, 144, 93, 145, 93, 146,
         94, 101, 94, 102, 94, 103, 94, 104, 94, 105,94, 106, 94, 107, 94, 108, 94, 109, 94, 110,
         94, 114, 94, 115, 94, 127, 94, 128, 94, 129,94, 130, 94, 131, 94, 132, 94, 133, 94, 134,
         94, 135, 94, 142, 94, 143, 94, 144, 94, 145,94, 146, 94, 147, 95, 101, 95, 102, 95, 110,
         95, 111, 95, 114, 95, 115, 95, 126, 95, 127,95, 128, 95, 129, 95, 130, 95, 131, 95, 132,
         95, 133, 95, 134, 95, 135, 95, 136, 95, 141,95, 142, 95, 143, 95, 146, 95, 147, 95, 148,
         96, 101, 96, 102, 96, 110, 96, 111, 96, 114,96, 115, 96, 126, 96, 127, 96, 135, 96, 136,
         96, 140, 96, 141, 96, 142, 96, 147, 96, 148,96, 149, 97, 101, 97, 102, 97, 110, 97, 111,
         97, 114, 97, 115, 97, 126, 97, 127, 97, 135,97, 136, 97, 139, 97, 140, 97, 141, 97, 148,
         97, 149, 97, 150, 98, 101, 98, 102, 98, 110,98, 111, 98, 114, 98, 115, 98, 125, 98, 126,
         98, 136, 98, 137, 98, 138, 98, 139, 98, 140,98, 149, 98, 150, 98, 151, 99, 101, 99, 102,
         99, 111, 99, 112, 99, 114, 99, 115, 99, 125,99, 126, 99, 136, 99, 137, 99, 138, 99, 139,
         99, 150, 99, 151
    };
     public boolean createEOP;
    private int filterarrar[][] =
    {
       {
       15, 10, 64
       },
    {
       200, 12, 128
    },
    {
        201, 12, 192
    },
    {
        91, 12, 256
    },
    {
        51, 12, 320
    },
    {
        52, 12, 384
    },
    {
        53, 12, 448
    },
    {
        108, 13, 512
    },
    {
```

```
    109, 13, 576
},
{
   74, 13, 640
},
{
   75, 13, 704
},
{
   76, 13, 768
},
{
   77, 13, 832
},
{
   114, 13, 896
},
{
   115, 13, 960
},
{
   116, 13, 1024
},
{
   117, 13, 1088
},
{
   118, 13, 1152
},
{
   119, 13, 1216
},
{
   82, 13, 1280
},
{
   83, 13, 1344
},
{
   84, 13, 1408
},
{
   85, 13, 1472
},
{
   90, 13, 1536
},
{
   91, 13, 1600
},
{
   100, 13, 1664
},
{
   101, 13, 1728
}
};
private int matharray[][] = {
{
   27, 5, 64
},
{
   18, 5, 128
},
{
   23, 6, 192
},
{
   55, 7, 256
},
{
   54, 8, 320
},
{
   55, 8, 384
},
{
   100, 8, 448
},
{
   101, 8, 512
},
{
   104, 8, 576
},
{
   103, 8, 640
```

```
},
{
   204, 9, 704
},
{
   205, 9, 768
},
{
   210, 9, 832
},
{
   211, 9, 896
},
{
   212, 9, 960
},
{
   213, 9, 1024
},
{
   214, 9, 1088
},
{
   215, 9, 1152
},
{
   216, 9, 1216
},
{
   217, 9, 1280
},
{
   218, 9, 1344
},
{
   219, 9, 1408
},
{
   152, 9, 1472
},
{
   153, 9, 1536
},
{
   154, 9, 1600
},
{
   24, 6, 1664
},
{
   155, 9, 1728
}
};
private int whitcodearray[][] = {
{
   55, 10, 0
},
{
   2, 3, 1
},
{
   3, 2, 2
},
{
   2, 2, 3
},
{
   3, 3, 4
},
{
   3, 4, 5
},
{
   2, 4, 6
},
{
   3, 5, 7
},
{
   5, 6, 8
},
{
   4, 6, 9
},
{
   4, 7, 10
},
```

```
{
   5, 7, 11
},
{
   7, 7, 12
},
{
   4, 8, 13
},
{
   7, 8, 14
},
{
   24, 9, 15
},
{
   23, 10, 16
},
{
   24, 10, 17
},
{
   8, 10, 18
},
{
   103, 11, 19
},
{
   104, 11, 20
},
{
   108, 11, 21
},
{
   55, 11, 22
},
{
   40, 11, 23
},
{
   23, 11, 24
},
{
   24, 11, 25
},
{
   202, 12, 26
},
{
   203, 12, 27
},
{
   204, 12, 28
},
{
   205, 12, 29
},
{
   104, 12, 30
},
{
   105, 12, 31
},
{
   106, 12, 32
},
{
   107, 12, 33
},
{
   210, 12, 34
},
{
   211, 12, 35
},
{
   212, 12, 36
},
{
   213, 12, 37
},
{
   214, 12, 38
},
{
   215, 12, 39
},
{
```

```
        108, 12, 40
    },
    {
        109, 12, 41
    },
    {
        218, 12, 42
    },
    {
        219, 12, 43
    },
    {
        84, 12, 44
    },
    {
        85, 12, 45
    },
    {
        86, 12, 46
    },
    {
        87, 12, 47
    },
    {
        100, 12, 48
    },
    {
        101, 12, 49
    },
    {
        82, 12, 50
    },
    {
        83, 12, 51
    },
    {
        36, 12, 52
    },
    {
        55, 12, 53
    },
    {
        56, 12, 54
    },
    {
        39, 12, 55
    },
    {
        40, 12, 56
    },
    {
        88, 12, 57
    },
    {
        89, 12, 58
    },
    {
        43, 12, 59
    },
    {
        44, 12, 60
    },
    {
        90, 12, 61
    },
    {
        102, 12, 62
    },
    {
        103, 12, 63
    }
    };
    private int codearray[][] = {
    {
        53, 8, 0
    },
    {
        7, 6, 1
    },
    {
        7, 4, 2
    },
    {
        8, 4, 3
    },
    {
        11, 4, 4
    },
```

```
{
   12, 4, 5
},
{
   14, 4, 6
},
{
   15, 4, 7
},
{
   19, 5, 8
},
{
   20, 5, 9
},
{
   7, 5, 10
},
{
   8, 5, 11
},
{
   8, 6, 12
},
{
   3, 6, 13
},
{
   52, 6, 14
},
{
   53, 6, 15
},
{
   42, 6, 16
},
{
   43, 6, 17
},
{
   39, 7, 18
},
{
   12, 7, 19
},
{
   8, 7, 20
},
{
   23, 7, 21
},
{
   3, 7, 22
},
{
   4, 7, 23
},
{
   40, 7, 24
},
{
   43, 7, 25
},
{
   19, 7, 26
},
{
   36, 7, 27
},
{
   24, 7, 28
},
{
   2, 8, 29
},
{
   3, 8, 30
},
{
   26, 8, 31
},
{
   27, 8, 32
},
{
   18, 8, 33
},
{
```

```
      19, 8, 34
   },
   {
      20, 8, 35
   },
   {
      21, 8, 36
   },
   {
      22, 8, 37
   },
   {
      23, 8, 38
   },
   {
      40, 8, 39
   },
   {
      41, 8, 40
   },
   {
      42, 8, 41
   },
   {
      43, 8, 42
   },
   {
      44, 8, 43
   },
   {
      45, 8, 44
   },
   {
      4, 8, 45
   },
   {
      5, 8, 46
   },
   {
      10, 8, 47
   },
   {
      11, 8, 48
   },
   {
      82, 8, 49
   },
   {
      83, 8, 50
   },
   {
      84, 8, 51
   },
   {
      85, 8, 52
   },
   {
      36, 8, 53
   },
   {
      37, 8, 54
   },
   {
      88, 8, 55
   },
   {
      89, 8, 56
   },
   {
      90, 8, 57
   },
   {
      91, 8, 58
   },
   {
      74, 8, 59
   },
   {
      75, 8, 60
   },
   {
      50, 8, 61
   },
   {
      51, 8, 62
   },
   {
      52, 8, 63
```

```
}
};
 private byte imagebit[];
 private int leftshift;
 private byte masking;
 private int imagearrcount;
 private byte bytearray[];
 public ImageToFaxEncoder()
 {
     debug = false;
     minBytesLine = 32;
     alignEOL = true;
     completeLine = true;
     lineWidth = 1728;
     pageLength = 2387;
     completePage = false;
     scaleImage = false;
     scaleFactor = 1;
     centerImage = false;
     whiteColor = -1;
     blackColor = 0;
     createEOP = false;
     imagebit = new byte[0x30d40];
     leftshift = 7;
     masking = 0;
     imagearrcount = 0;
     bytearray = new byte[0];
 }
 private void setBitMap(int b)
 {
     byte bitMask = 0;
     bitMask = (byte)(1 << leftshift);
     if(b == 1)
         masking = (byte)(masking | bitMask);
        if(debug)
        System.out.print("".concat(String.valueOf(String.valueOf(b))));
        leftshift--;
        if(leftshift < 0)
        {
            leftshift = 7;
            imagebit[imagearrcount] = masking;
            imagearrcount++;
            masking = 0;
        }
    }
    private void setBitMask(int code, int bits)
    {
      int mask = 0;
      for(mask = 1 << bits - 1; mask != 0; mask >>= 1)
      if((code & mask) > 0)
         setBitMap(1);
      else
         setBitMap(0);
      if(debug)
               System.out.print(" ");
    }
   private void checkBitMap()
   {
      if(alignEOL)
      while(leftshift != 3)
      setBitMap(0);
      for(int h = 0; h < 11; h++)
      setBitMap(0);
      setBitMap(1);
    }
    private void _$41684()
    {
         for(int h = 0; h < 6; h++)
         checkBitMap();
    }
   public void addFiller(int runl, boolean inWhite)
   {
    int codes[][];
    int code[];
    if(runl >= 64)
    {
         if(inWhite)
         codes = matharray;
         else
         codes = filterarrar;
         code = codes[(int)Math.floor(runl / 64 - 1)];
         runl -= code[2];
         setBitMask(code[0], code[1]);
     }
     if(inWhite)
         codes = codearray;
     else
         codes = whitcodearray;
```

```
        code = codes[runl];
        setBitMask(code[0], code[1]);
    }
    public byte[] encodeImage(Image i)
    {
     int scale = scaleFactor;
     int currentPageRow = 0;
     if(scaleImage)
     {
          int w = i.getWidth(null);
          int h = i.getHeight(null);
          for(int j = 2; j < 10 && j * w <= lineWidth && j * h <= pageLength; j++)
          scale = j;
       }
        imagearrcount = 0;
        leftshift = 7;
        masking = 0;
        checkBitMap();
        int iw = i.getWidth(null);
        int ih = i.getHeight(null);
        int pix[] = new int[iw * ih];
        String s = "";
        String c = "";
        int runl = 0;
        int code[] = null;
        int lineBytesOffset = 0;
        PixelGrabber pg = new PixelGrabber(i, 0, 0, iw, ih, pix, 0, iw);
        try
        {
            pg.grabPixels();
        }
        catch(Exception e)
        {
            System.err.println("".concat(String.valueOf(String.valueOf(e.getMessage()))));
        }
         int bp = 1;
         if(bp == whiteColor)
         bp = 0;
         for(int h = 0; h < EOLimagearray.length; h += 2)
         pix[EOLimagearray[h] * iw + EOLimagearray[h + 1]] = bp;
         for(int row = 0; row < ih; row++)
         {
               for(int sc = 1; sc <= scale; sc++)
               {
                     currentPageRow++;
                     int ocol = 0;
                     int totalRow = 0;
                     lineBytesOffset = imagearrcount;
                     for(int col = 0; col < iw;)
                     {
                          runl = 0;
                        if(col == 0 && completeLine && centerImage)
                        {
                           totalRow = (lineWidth - iw * scale) / 2;
                           runl = totalRow;
                        }
                        while(col < lineWidth && col < iw && pix[row * iw + col] == whiteColor)
                        {
                            runl += scale;
                            col++;
                            totalRow += scale;
                        }
                        if(col >= iw && completeLine)
                        {
                           runl += lineWidth - totalRow;
                           totalRow = lineWidth;
                       }
                       if(runl >= 64)
                       {
                           code = matharray[(int)Math.floor(runl / 64 - 1)];
                           runl -= code[2];
                           if(debug)
                           System.out.println(String.valueOf(String.valueOf(
                           (new StringBuffer("White length=")).append(code[2]).append
                           (" code=").append(code[0]).append(" bits=").append(code[1]))));
                           setBitMask(code[0], code[1]);
                       }
                       code = codearray[runl];
                       if(debug)
                        System.out.println(String.valueOf(String.valueOf(
                        (new StringBuffer("White length=")).append(runl).append
                        (" code=").append(code[0]).append(" bits=").append(code[1]))));
                        setBitMask(code[0], code[1]);
                        if(col >= lineWidth || col >= iw)
                        break;
                        runl = 0;
                        while(col < lineWidth && col < iw && pix[row * iw + col] != whiteColor)
                       {
```

```
                            runl += scale;
                            col++;
                            totalRow += scale;
                        }
                        if(runl >= 64)
                        {
                            code = filterarrar[(int)Math.floor(runl / 64 - 1)];
                            runl -= code[2];
                            if(debug)
                            System.out.println(String.valueOf(String.valueOf(
                            (new StringBuffer("Black length=")).append(code[2]).append
                            (" code=").append(code[0]).append(" bits=").append(code[1]))));
                            setBitMask(code[0], code[1]);
                        }
                        code = whitcodearray[runl];
                        if(debug)
                         System.out.println(String.valueOf(String.valueOf(
                         (new StringBuffer("Black length=")).append(runl).append
                         (" code=").append(code[0]).append(" bits=").append(code[1]))));
                         setBitMask(code[0], code[1]);
                    }
                    if(completeLine && totalRow < lineWidth)
                    {
                        runl = lineWidth - totalRow;
                        if(runl >= 64)
                        {
                            code = matharray[(int)Math.floor(runl / 64 - 1)];
                            runl -= code[2];
                            setBitMask(code[0], code[1]);
                        }
                        code = codearray[runl];
                        setBitMask(code[0], code[1]);
                    }
                    while(minBytesLine > imagearrcount - lineBytesOffset)
                    {
                        int n = 0;
                        while(n < 8)
                        {
                            setBitMap(0);
                            n++;
                        }
                    }
                    checkBitMap();
                    if(debug)
                     System.out.println(" EOL");
                }
            }
             if(completePage)
             for(; currentPageRow < pageLength; currentPageRow++)
             {
             runl = lineWidth;
             if(runl >= 64)
             {
                code = matharray[(int)Math.floor(runl / 64 - 1)];
                runl -= code[2];
                setBitMask(code[0], code[1]);
             }
             code = codearray[runl];
             setBitMask(code[0], code[1]);
             checkBitMap();
             }
            if(createEOP)
             _$41684();
            if(leftshift != 7)
            {
                imagebit[imagearrcount] = masking;
                imagearrcount++;
            }
            byte result[] = new byte[imagearrcount];
            for(int h = 0; h < imagearrcount; h++)
            result[h] = imagebit[h];
             return result;
        }
}
```

Download this listing.

In the above listing, the ImageToFaxEncoder class encodes the fax document. The various methods defined in the ImageToFaxEncoder.java file are:

- `setBitMap()`:Sets the bitmap for the image that the Fax application sends as a fax document.
- `setBitMask()`:Sets the mask bit for the image that the Fax application sends as a fax document.
- `checkBitMap()`:Checks the bitmap for the image sent as a fax.
- `_$41684()`:Invokes the checkBitMap() method.

- `addFiller()`:Adds the file filter for the image to open only image files.
- `encodeImage()`: Encodes the image that the Fax application sends as a fax.

PREVIOUS NEXT

Converting the Encoded Fax Image into Frames

The LCFrame.java file converts the encoded image of the document sent using the Fax application into frames.

Listing 6-9 shows the LCFrame.java file:

Listing 6-9: The LCFrame.java File

```
public class LCFrame
{
    byte bytes[];
    int bytesCount;
    public LCFrame(byte b[], int len)
    {
        bytes = new byte[256];
        bytesCount = 0;
        bytes = b;
        bytesCount = len;
    }
    public LCFrame()
    {
        bytes = new byte[256];
        bytesCount = 0;
        bytes[0] = -1;
        bytesCount++;
        bytes[1] = 3;
        bytesCount++;
        bytes[2] = 0;
        bytesCount++;
    }
    public void addByte(byte b)
    {
        bytes[bytesCount++] = b;
    }
    public void setLast(boolean l)
    {
        if(l)
            bytes[1] = 19;
        else
            bytes[1] = 3;
    }
    public int getFrameType()
    {
        if(bytesCount > 2)
        {
            int b = bytes[2];
            if(b < 0)
                b = 256 + b;
            return b;
        }
      else
      {
          return 0;
      }
   }
    public void setFrameType(byte t)
    {
        if(bytesCount > 2)
            bytes[2] = t;
    }
    public boolean isLast()
    {
        if(bytesCount > 1)
            return bytes[1] == 19;
        else
            return false;
    }
    public byte[] getData()
    {
        if(bytesCount <= 5)
            return null;
        byte r[] = new byte[bytesCount - 5];
        for(int h = 0; h < r.length; h++)
            r[h] = bytes[h + 3];
        return r;
    }
    public byte[] getRawData()
    {
        byte r[] = new byte[bytesCount];
        for(int h = 0; h < r.length; h++)
            r[h] = bytes[h];
        return r;
    }
}
```

Download this listing.

In the above listing, the LCFrame class converts a fax image to frames. The various methods defined in the LCFrame.java file are:

- `addByte()`: Adds a byte to the fax document.
- `getFrameType()`: Determines the type of frame that the Fax application sends.
- `setFrameType()`: Sets the frame type for the image that the Fax application sends.
- `isLast()`:Determines whether the current data byte is the last byte or not.
- `getData()`:Retrieves the data that is converted into frames using the LCFrame class.
- `getRawData()`:Retrieves the data that the Fax application sends in the form of frames.

PREVIOUS NEXT

Sending the Fax

The SendingFax.java file sends the fax to the number specified in the Fax Number text box of the Fax Application window.

Listing 6-10 shows the SendingFax.java file:

Listing 6-10: The SendingFax.java File

```
/*Imports required java.awt.image class*/
import java.awt.Image;
/*Imports required java.i/o package classes*/
import java.io.*;
/*Imports required java.text.DateFormat class*/
import java.text.DateFormat;
/*
Imports required java.text.SimpleDateFormat class
*/
import java.text.SimpleDateFormat;
/*Imports required java.util.Calendar class*/
import java.util.Calendar;
import java.util.Date;
import javax.comm.*;
/*
class SendingFax - This class is used to send commands to modem to send fax
Constructor:
   SendingFax-This constructor creates GUI.
Methods:
   openForFax - This method is used to send fax.
actionPerformed - This method describes which action will be performed on which component.
main - This method creates the main window of the application and displays all the components.*/
class SendingFax
{
   /*Declare objects of String class.*/
      public String flowcontrolin;
      protected String modemresponse;
      public String dialingmode;
      public String noecho;
      public String ownid;
      public String logStr;
      public String initialcommands[];
      public String port;
      public String initString;
      public String faxFile;
      public String lastError;
      public String resetcommand;
      public int flowcontrol;
      public boolean bitorder;
      public boolean modemFBOR;
      public int faxclass;
      public boolean log;
      public boolean debug;
      public PreparedFaxDoc producer;
      protected int pages;
      protected ImageToFaxEncoder encoder;
      public PortFax listener;
      protected boolean V27Supported;
      protected boolean V29Supported;
      protected boolean V17Supported;
      public int MPSEOPdelay;
      public int buffersize;
      long startTime;
      long tout;
      /*
      Declare objects of CommPortIdentifier class*/
      CommPortIdentifier commportid;
      /*Declare objects of SerialPort class*/
      SerialPort serialPort;
      /*Declare objects of OutputStream class*/
      OutputStream outputStream;
      /*Declare objects of InputStream class*/
      InputStream inputStream;
      protected byte reverseBytes[];
      boolean isClass1;
      boolean isClass2;
      boolean isClass20;
      protected static double TRAINING_SECONDS = 1.5D;
      protected ModemCapabilities cap;
      public int maxRetries;
      protected static int brClass2_to_Class1[] = {24, 48, 72, 96, 121, 145};
      protected static String brClass2_to_Class1V[] = {"V27", "V27", "V29", "V29", "V17", "V17"};
       public int bitrate;
       public int resolution;
       public static final int RESOLUTION_NORMAL = 0;
       public static final int RESOLUTION_FINE = 1;
```

```
 public int timeout;
 public int resetdelay;
 public int lastresponse;
 public int pagecode;
 protected int postcode;
 public int hangcode;
 public SendingFax()
{
      reverseBytes = new byte[256];
      port = "COM1";
      isClass1 = false;
      isClass2 = false;
      isClass20 = false;
      initString = "ATV1Q0";
      maxRetries = 3;
      bitrate = 3;
      resolution = 1;
      faxFile = "tmpFax";
      lastError = "";
      resetcommand = "ATZ";
      timeout = 30;
      resetdelay = 0;
      lastresponse = 0;
      pagecode = -1;
      postcode = -1;
      hangcode = -1;
      flowcontrol = 0;
      modemresponse = "";
      noecho = "ATE0";
      flowcontrolin="";
      bitorder = false;
      modemFBOR = true;
      dialingmode = "";
      faxclass = 2;
      ownid = "12345";
      log = true;
      debug = true;
      logStr = "";
      initialcommands = new String[0];
      pages = 0;
      encoder = null;
      listener = null;
      V27Supported = true;
      V29Supported = true;
      V17Supported = false;
      MPSEOPdelay = 8;
      buffersize = 0x3d090;
      startTime = 0L;
      tout = 0L;
      /*
      Initialize object of ImageToFaxEncoder class.
      */
      encoder = new ImageToFaxEncoder();
      encoder.minBytesLine = 0;
      for(int i = 0; i < 256; i++)
     {
         byte b = (byte)i;
         byte bi = 0;
         if((b & 0x80) > 0)
             bi |= 1;
         if((b & 0x40) > 0
            bi |= 2;
         if((b & 0x20) > 0)
             bi |= 4;
         if((b & 0x10) > 0)
             bi |= 8;
         if((b & 8) > 0)
             bi |= 0x10;
         if((b & 4) > 0)
             bi |= 0x20;
         if((b & 2) > 0)
             bi |= 0x40;
         if((b & 1) > 0)
             bi |= 0x80;
         reverseBytes[i] = bi;
     }
}
/*
openForFax - This method is called to open and prepare a port to send fax.
Parameters: preparedfaxdoc - Object of PreparedFaxDoc class
Return Value:boolean
*/
public boolean openForFax(PreparedFaxDoc preparedfaxdoc)
{
/*createFaxFiles method is called.*/
     if(!createFaxFiles(preparedfaxdoc))
     {
         lastError = "Could not create fax files.";
```

```
            return false;
        }
        if(listener != null)
         listener.faxProgress(1, 0);
         /*
         Initialize object of CommPortIdentifier.
         */
        try
        {
            commportid = ommPortIdentifier.getPortIdentifier(port);
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            lastError = "Could not find port ".concat(String.valueOf(String.valueOf(port)));
            boolean flag = false;
            return flag;
    }
    if(commportid == null)
    {
        lastError = "Could not find port ".concat(String.valueOf(String.valueOf(port)));
        return false;
    }
    if(commportid.getPortType() != 1)
    {
        lastError = String.valueOf(String.valueOf(
        (new StringBuffer("")).append(port).append(" is not a serial port")));
        return false;
    }
    /*Open the serial port.*/
    try
    {
        serialPort = (SerialPort)commportid.open("Fax Application", 2000);
    }
    catch(PortInUseException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not open port ".concat(String.valueOf(String.valueOf(port)));
        boolean flag1 = false;
        return flag1;
    }
    /*Sets port receive time.*/
    try
    {
        serialPort.enableReceiveTimeout(1000);
    }
    catch(Exception exception) { }
/*Gets input and output stream.*/
    try
    {
        outputStream = serialPort.getOutputStream();
        inputStream = serialPort.getInputStream();
    }
    catch(IOException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not get streams";
        boolean flag2 = false;
        return flag2;
    }
/*Sets serial port parameter.*/
    try
    {
        serialPort.setSerialPortParams(19200, 8, 1, 0);
    }
    catch(UnsupportedCommOperationException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not configure port";
        boolean flag3 = false;
        return flag3;
    }
   if(listener != null)
   listener.faxProgress(2, 0);
   /*Send reset command to modem.*/
   if(!sendATCommand(resetcommand))
   {
        lastError = "Could not reset modem with ATZ";
        return false;
    }
  if(resetdelay > 0)
  /*softDelay method is called.*/
  softDelay(resetdelay);
  if(flowcontrol == 2 && flowcontrolin.length()>0)
  sendATCommand(flowcontrolin);
  if(flowcontrol == 1 &&   flowcontrolin.length()>0)
  sendATCommand(flowcontrolin);
  if(flowcontrol == 0 && flowcontrolin.length()>0)
```

```
sendATCommand(flowcontrolin);
try
{
   if(flowcontrol == 2)
      serialPort.setFlowControlMode(2);
      if(faxclass != 1 && flowcontrol == 1)
         serialPort.setFlowControlMode(8);
      if(flowcontrol == 0)
          serialPort.setFlowControlMode(0);
  }
  catch(Exception e)
  {
      System.err.println(e.getMessage());
  }
  /*Add flow control type in log file.*/
  if(log)
  {
      if((serialPort.getFlowControlMode() & 8) > 0) addLogLn("FLOWCONTROL_XONXOFF_OUT");
      if((serialPort.getFlowControlMode() & 4) > 0) addLogLn("FLOWCONTROL_XONXOFF_IN");
      if((serialPort.getFlowControlMode() & 1) > 0) addLogLn("FLOWCONTROL_RTSCTS_IN");
      if((serialPort.getFlowControlMode() & 2) > 0) addLogLn("FLOWCONTROL_RTSCTS_OUT");
  }
  /*Send echo command to modem.*/
  sendATCommand(noecho);
  sendATCommand(initString);
  /*queryCapabilities method is called.*/
  queryCapabilities();
  if(isClass1 && faxclass == 1)
  {
     /*
      Send service class selection command
      */
      if(!sendATCommand("AT+FCLASS=1"))
      {
          lastError = "Could not set class 1";
          return false;
      }
   /*queryBR method is called.*/
      queryBR();
  }
  if(isClass2 && faxclass == 2)
  {
      /*
      Send service class selection command
      */
      if(!sendATCommand("AT+FCLASS=2"))
      {
          lastError = "Could not set class 2";
          return false;
      }
      if(debug)
      /*Send HDLC frame tracing command.*/
      sendATCommand("AT+FBUG=1");
  }
  if(isClass20 && faxclass == 20)
  {
      /*Send service class selection command*/
      if(!sendATCommand("AT+FCLASS=2.0"))
      {
         lastError = "Could not set class 2.0";
         return false;
      }
      sendATCommand("AT+FNR=1,1,1,0");
      if(debug)
          sendATCommand("AT+FBU=1");
  }
  if(modemFBOR)
  {
      if(faxclass == 2)
      {
         if(!bitorder)
         /*
         The FBOR command determines the order in which data bits are transmitted
         between the DTE and the modem and between the modem and the PSTN.
         +FBOR=0 selects direct bit order where the first bit of a byte sent to the
         modem is the first bit sent down the PSTN.
         +FBOR=1 selects reverse bit order where the first bit of a byte sent to the
         modem is the last bit sent down the PSTN.
         */
         sendATCommand("AT+FBOR=1");
         if(bitorder)
         sendATCommand("AT+FBOR=0");
      }
      if(faxclass == 20)
      {
         /*
         Send command to setup data bit order.
         */
```

```
            if(!bitorder)
            sendATCommand("AT+FBO=1");
            if(bitorder)
            sendATCommand("AT+FBO=0");
          }
      }
      if(ownid.length() > 0)
      {
          /*Send Local ID string command.*/
          if(faxclass == 20)
          sendATCommand(String.valueOf(String.valueOf((new StringBuffer("AT+FLI=\"")).append(ownid
          if(faxclass == 2)
          /*
          This command is used to set the local identifying information stored within
          the modem. This allows the remote system to identify the calling station.
          The string may be up to 20 ASCII characters in length.
          */sendATCommand(String.valueOf(String.valueOf(
          (new StringBuffer("AT+FLID=\"")).append(ownid).append("\""))));
      }
       /*
       Send command for modem capabilities parameters.
       */
       if(faxclass == 20)
       sendATCommand(String.valueOf(String.valueOf(
       (new StringBuffer("AT+FCC=")).append(resolution).append(",").append
       (bitrate).append(",0,2,0,0,0,0"))));
       if(faxclass == 2)
       sendATCommand(String.valueOf(String.valueOf(
       (new StringBuffer("AT+FDCC=")).append(resolution).append(",").append
       (bitrate).append(",0,2,0,0,0,0"))));
       for(int i = 0; i < initialcommands.length; i++)
       {
           System.out.println("");
            sendATCommand(initialcommands[i]);
       }
       return true;
 }
/*
createFaxFiles - This method is called to create image of editpane content.
Parameters: p - Object of PreparedFaxDoc class
Return Value:boolean
*/
protected boolean createFaxFiles(PreparedFaxDoc p)
 {
   if(faxclass == 1)
         bitorder = true;
     if(p != null)
         producer = p;
     if(producer == null)
         return false;
     if(listener != null)
   /*faxProgress method is called.*/
         listener.faxProgress(6, 0);
        pages = 0;
        encoder.createEOP = faxclass == 20;
     if(faxclass == 1)
         encoder.createEOP = false;
      Image page = null;
   for(page = producer.getFaxPage(pages); page != null; page = producer.getFaxPage(pages))
   {
      byte pageBytes[] = encoder.encodeImage(page);
      if(pageBytes != null)
      {
      /*Write string in the file.*/
      writeFile(String.valueOf(String.valueOf(
      (new StringBuffer(String.valueOf(String.valueOf(faxFile)))).append(".").append
      (pages))), pageBytes, pageBytes.length);
      pages++;
      }
   }
   return true;
 }
/*
setPortName - This method is called to set the port name.
Parameters: port - Object of String class
Return Value: NA
*/
 public void setPortName(String port)
 {
  this.port = port;
 }
/*
setInitString - This method is called to set the modem initial string.
Parameters: s - Object of String class
Return Value: NA
*/
 public void setInitString(String s)
```

```
 {
  initString = s;
 }
/*
writeFile - This method is called to write string in the file.
Parameters: f - Object of String class,b[] - array of bytes,count - value of integer type.
Return Value:boolean
*/
    protected boolean writeFile(String f, byte b[], int count)
    {
     try
     {
         FileOutputStream fo = new FileOutputStream(f);
         fo.write(b, 0, count);
         fo.close();
     }
     catch(Exception e)
     {
         System.err.println("Error writing to file
         ".concat(String.valueOf(String.valueOf(e.getMessage()))));
         boolean flag = false;
         return flag;
     }
     return true;
 }
/*
sendATCommand - This method is called to send the command to the  modem.
Parameters: messageString - Object of String class
Return Value:boolean
*/
protected boolean sendATCommand(String messageString)
{
    if(messageString.trim().length() == 0)
    return true;
    if(sendATCommandnoWait(messageString))
    return waitFor("OK");
    else
    return false;
 }
/*
sendATCommandnoWait - This method is called to send the wait command to the modem.
Parameters: messageString - Object of String class
Return Value:boolean
*/
protected boolean sendATCommandnoWait(String messageString)
 {
     char cr = '\r';
     try
     {
         if(log)
         {
                addLogLn("");
                addLogLn("Command out.");
         }
         if(log)
         addLog(messageString);
         messageString = String.valueOf(messageString) + String.valueOf(cr);
         /*Write output stream.*/
         outputStream.write(messageString.getBytes());
     }
     catch(IOException e)
     {
         System.err.println(e.getMessage());
            boolean flag = false;
         return flag;
     }
     return true;
  }
   /*
   addLogLn - This method is called to display the fax status.
   Parameters: s - Object of String class
   Return Value: NA
   */
    protected void addLogLn(String s)
    {
      logStr = String.valueOf(String.valueOf(
      (new StringBuffer(String.valueOf(String.valueOf(logStr)))).append("\n").append(s)));
      if(debug)
      System.out.println(s);
    }
      /*
      softDelay - This method is called to display the fax status.
      Parameters: t - integer type variable.
      Return Value: NA
      */
    protected void softDelay(int t)
    {
```

```
     try
     {
         Thread.currentThread();
         Thread.sleep(t);
     }
     catch(Exception exception) { }
 }
/*
queryCapabilities - This method is called to set the fax class.
Parameters: NA
Return Value: Nected void queryCapabilities()
{A
*/
prot
   sendATCommand("AT+FCLASS=?");
   if(modemresponse.indexOf("1") > 0)
      isClass1 = true;
   if(modemresponse.indexOf("2") > 0)
      isClass2 = true;
   if(modemresponse.indexOf("2.0") > 0)
      isClass20 = true;
  }
   /*
   queryBR - This method is called to set the modem variables.
   Parameters: NA
   Return Value: NA
   */
 protected void queryBR()
{
   if(isClass1)
   {
      sendATCommand("AT+FTM=?");
      if(modemresponse.indexOf("48") > 0)
         V27Supported = true;
      if(modemresponse.indexOf("96") > 0)
         V29Supported = true;
      if(modemresponse.indexOf("97") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("98") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("145") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("146") > 0)
         V17Supported = true;
   }
}
/*
waitFor - This method is used to invoke waitForOK method.
Parameters: resp - Object of String class.
Return Value:boolean
*/
 public boolean waitFor(String resp)
 {
   return waitForOK(resp, true);
 }
/*
waitForOK - This method is used to invoke waitForOK method.
Parameters:  resp - object of String class, wOK - boolean type variable.
Return Value:boolean
 */
protected boolean waitForOK(String resp, boolean wOK)
 {
     return waitForOK(resp, wOK, timeout);
 }
   protected void addLog(String s)
    {
        logStr = String.valueOf(logStr) + String.valueOf(s);
        if(debug)
        System.out.print(s);
    }
    /*
    waitForOK - This method is called to call waitForOK method.
    Parameters: resp - object of String class, wOK - boolean type variable, ptout - integer type
    Return Value:boolean
    */
    protected boolean waitForOK(String resp, boolean wOK, int ptout)
    {
        String r = "";
        String line = "";
        modemresponse = "";
        Calendar cal = Calendar.getInstance();
        long startTime = cal.getTime().getTime();
        try
        {
            if(log)
            {
                   addLogLn("");
                   addLog("Response: ");
```

```
    }
int c = inputStream.read();
r = String.valueOf(r) + String.valueOf((char)c);
do
{
   long now = Calendar.getInstance().getTime().getTime();
   Calendar cale = Calendar.getInstance();
   cale.add(6, 7);
   String dateStr = (new SimpleDateFormat("ddmmyyyy")).format(cale.getTime());
   if(startTime < now && now - startTime > (long)(ptout * 1000))
   {
      timeout = 9;
      lastError = "Timeout waiting for response";
      boolean flag1 = false;
      return flag1;
    }
    if(c == 13)
    {
         if(line.indexOf("+FHNG:") >= 0 || line.indexOf("+FHS:") >= 0 ||
         line.indexOf("+FHG:") >= 0)
         hangcode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FPS:") >= 0 || line.indexOf("+FPTS:") >= 0)
         pagecode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FET:") >= 0)
         postcode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FDSC:") >= 0)
        {
            String capStr = line.substring(line.indexOf(":") + 1, line.length());
            ModemCapabilities tmpCap = new ModemCapabilities();
            tmpCap.decodeCapabilitiesClass2(capStr);
         }
         if(line.indexOf("+FCON") >= 0 || line.indexOf("+FCO") >= 0)
         lastresponse = 5;
         if(line.indexOf("CONNECT") >= 0 && faxclass == 1)
         lastresponse = 5;
         if(line.indexOf("ERROR") >= 0 || line.indexOf("FCERROR") >= 0)
         {
               lastresponse = 1;
                  boolean flag2 = false;
               return flag2;
         }
         if(line.indexOf("BUSY") >= 0)
         {
               lastresponse = 8;
               lastError = "Line busy";
               boolean flag3 = false;
               return flag3;
         }
         if(line.indexOf("NO DIALTONE") >= 0)
         {
               lastresponse = 7;
               lastError = "No dial tone";
               boolean flag4 = false;
               return flag4;
          }
         if(line.indexOf("NO CARRIER") >= 0 && resp.compareTo("NO CARRIER") != 0)
         {
               lastresponse = 6;
               lastError = "No carrier";
               boolean flag5 = false;
               return flag5;
         }
          line = "";
    }
    if(log)
   addLogCh(c);
   if(c == 13 && r.indexOf(resp) >= 0 && (r.indexOf("OK") >= 0 || !wOK))
   break;
   if(c == 13 && r.indexOf("+FHNG") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag6 = false;
        return flag6;
    }
   if(c == 13 && r.indexOf("+FHG") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag7 = false;
        return flag7;
    }
   if(c == 13 && r.indexOf("+FHS") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag8 = false;
        return flag8;
    }
    if(r.indexOf("ERROR") >= 0)
    {
```

```
                    boolean flag9 = false;
                   return flag9;
                }
               c = inputStream.read();
               r = String.valueOf(r) + String.valueOf((char)c);
               modemresponse = String.valueOf(modemresponse) + String.valueOf((char)c);
               if(c != 10 && c != 13)
               line = String.valueOf(line) + String.valueOf((char)c);
            }
            while(true);
            if(log)
                addLogLn("");
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            boolean flag = false;
            return flag;
        }
        return true;
    }
/*
addLogCh - This method is called to print log document.
Parameters: c - integer type variable.
Return Value: NA
*/
protected void addLogCh(int c)
    {
        String cs = "";
        if(c < 32)
         cs = String.valueOf(String.valueOf((new StringBuffer("<")).append(c).append(">")));
        else
         cs = "".concat(String.valueOf(String.valueOf((char)c)));
         logStr = String.valueOf(logStr) + String.valueOf(cs);
        if(debug)
         System.out.print(cs);
    }
    /*
   close - This method is called to close input and output stream.
   Parameters: NA
   Return Value:boolean
   */
    public boolean close()
    {
        for(int c = 0; c < pages; c++)
        {
            File file;
            try
            {
               file = new File(String.valueOf(String.valueOf(
               (new StringBuffer(String.valueOf(String.valueOf(faxFile))))
               .append(".").append(c))));
            }
            catch(Exception exception) { }
        }
        try
        {
            inputStream.close();
            outputStream.close();
            serialPort.close();
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            boolean flag = false;
            return flag;
        }
        return true;
    }
/*
sendFax - This method is called to send the fax
Parameters: destId - Object of String class.
Return Value:boolean
 */
 public boolean sendFax(String destId)
    {
     hangcode = -1;
     boolean resendPage = false;
     int retry = 0;
     String r = "";
     if(listener != null)
         listener.faxProgress(3, 0);
     if(connect(destId))
     {
         if(faxclass == 1)
         {
             if(!phaseB())
                 return false;
```

```
    if(!sendTSI())
        return false;
}
int pageCount = 0;
boolean retrain = true;
System.out.println("send");
do
{
    if(pageCount >= pages)
        break;
    if(listener != null)
   listener.faxProgress(4, pageCount);
    boolean sendingUserFile = false;
    byte pageBytes[];
    if((new File("send.g3")).exists())
    {
      pageBytes = readFile("send.g3");
      sendingUserFile = true;
      System.out.println("** SENDING USER FILE *** send.g3");
    }
   else
    {
         pageBytes = readFile(String.valueOf(String.valueOf(
         (new StringBuffer(String.valueOf(String.valueOf(faxFile))))
         .append(".").append(pageCount))));
    }
    if(pageBytes == null)
    {
      lastError = "Could not read fax file.";
      return false;
    }
    if(faxclass == 1 && retrain)
    {
        if(!sendDCS())
            return false;
        if(!doTraining())
            return false;
    }
    retrain = false;
   if((faxclass == 2 || faxclass == 20) && !sendATCommandnoWait("AT+FDT"))
   {
      lastError = "Could not start phase C";
      return false;
    }
    if(faxclass == 20)
       waitForOK("CONNECT", false);
    if(faxclass == 2)
    {
         int previousFlowControl = serialPort.getFlowControlMode();
         try
           {
               serialPort.setFlowControlMode(0);
           }
            catch(Exception exception) { }
           try
           {
               if(log)
               addLog("Response: ");
               r = "";
               int c = inputStream.read();
               r = String.valueOf(r) + String.valueOf(c);
               do
               {
                   if(c == 17)
                   break;
                   c = inputStream.read();
                   if(log)
                   addLogCh(c);
                   r = String.valueOf(r) + String.valueOf(c);
                   if(c == 13 && r.indexOf("OK") >= 0)
                   break;
                   if(c == 17 && log)
                   addLogLn("XON received");
               }
               while(true);
               if(r.indexOf("+FHNG") >= 0)
               {
                  lastresponse = 2;
                  lastError = "Remote closed connection";
                  boolean flag = false;
                  return flag;
               }
               if(r.indexOf("+FHS") >= 0)
               {
                  lastresponse = 2;
                  lastError = "Remote closed connection";
                  boolean flag1 = false;
                  return flag1;
```

```
                }
            }
            catch(Exception e)
            {
                System.err.println(e.getMessage());
                lastError = "Error reading (wait for XON)";
                boolean flag2 = false;
                return flag2;
            }
            try
            {
                serialPort.setFlowControlMode(previousFlowControl);
            }
            catch(Exception exception1) { }
        }
        pagecode = -1;
        if(faxclass == 1)
        {
            if(!sendDataClass1(pageBytes, sendingUserFile))
            {
                lastError = "Could not send page bytes";
                return false;
            }
            byte eop[] = {0, 8, -128, 0, 8, -128, 0, 8, -128
            };
            sendBytes(eop, eop.length);
            byte rtc[] = {16, 3};
            sendBytes(rtc, rtc.length);
    }
    else
    if(!sendData(pageBytes))
    {
        lastError = "Could not send page bytes";
        return false;
    }
    if(faxclass == 20)
    {
        if(pageCount == pages - 1)
        {
            byte rtc[] = {16, 46};
            sendBytes(rtc, rtc.length);
        }
        else
        {
            byte rtc[] = {16, 44};
            sendBytes(rtc, rtc.length);
        }
        sendATCommand("AT+FPS?");
    }
    if(faxclass == 2)
    {
        byte rtc[] = {16, 3};
        sendBytes(rtc, rtc.length);
        try
        {
            Thread.currentThread();
            Thread.sleep(500L);
        }
        catch(Exception exception2) { }
        if(pageCount == pages - 1)
            sendATCommand("AT+FET=2");
        else
            sendATCommand("AT+FET=0");
        if(pagecode == -1)
            sendATCommand("AT+FPTS?");
    }
    resendPage = false;
    if(faxclass == 1)
    {
        if(!sendATCommand("AT+FTS=".concat(String.valueOf(String.valueOf(MPSEOPdelay)))))
            return false;
        int PPMretry = 1;
        LCFrame rsp = null;
        do
        {
            if(PPMretry > 3)
                break;
            if(!sendClass1FET(pageCount == pages - 1))
            return false;
            if(!sendATCommandnoWait("AT+FRH=3"))
            return false;
            rsp = waitForFrame(3000);
            if(rsp != null)
            break;
            lastError = "";
            PPMretry++;
        }
        while(true);
```

```
                    if(rsp == null)
                    {
                          lastError = "Response to MPS/EOP not received after 3 retries.";
                          return false;
                      }
                      if(rsp.getFrameType() == 44)
                      {
                          lastError = "Procedure interrupted";
                          return false;
                      }
                      if(rsp.getFrameType() == 172)
                      {
                          lastError = "Procedure interrupted";
                          return false;
                      }
                      if(rsp.getFrameType() == 140 && log)
                      addLogLn("Positive message confirmation.");
                      if(pageCount == pages - 1 && rsp.getFrameType() == 204)
                      {
                          retrain = true;
                          if(log)
                          addLogLn("Retrain positive.");
                      }
                      if(rsp.getFrameType() == 76)
                      {
                             retrain = true;
                          if(retry < maxRetries)
                          {
                                resendPage = true;
                                retry++;
                          }
                      }
                      if(rsp.getFrameType() == 250)
                      {
                          lastError = "Disconnect frame received";
                          hangcode = 20;
                          break;
                      }
                  }
                  if(faxclass == 2 || faxclass == 20)
                  switch(pagecode)
                  {
                     case -1:
                     case 0:
                     case 1:
                     default:
                     break;
                     case 2:
                     if(retry < maxRetries)
                     {
                         resendPage = true;
                         retry++;
                     }
                     break;
                  }
                  if(!resendPage)
                  pageCount++;
              }
              while(hangcode < 0);
              if(listener != null)
              listener.faxProgress(5, 0);
              if(faxclass == 1 && sendClass1Disconnect())
              hangcode = 0;
              hangup();
              if(hangcode == 0)
              return true;
          }
          return false;
      }
      /*
      connect - This method is called to connect the modem.
      Parameters: destId - Object of String class.
      Return Value:boolean
      */
      protected boolean connect(String destId)
       {
         String dialMode = "P";
         if(dialingmode=="Tone")
            dialMode = "T";
         if(faxclass == 1)
         {
            if(!sendATCommandnoWait(String.valueOf(String.valueOf(
            (new StringBuffer("ATD")).append(dialMode).append(destId)))))
            return false;
            waitForOK("CONNECT", false);
         }
         else
         if(!sendATCommand(String.valueOf(String.valueOf(
```

```
        (new StringBuffer("ATD")).append(dialMode).append(destId)))))
        return false;
        return lastresponse == 5;
      }
      protected boolean phaseB()
      {
          LCFrame f = null;
          boolean haveDIS = false;
          do
          {
              f = waitForFrame(10000);
              if(f == null)
                    return false;
              if(!waitForOK("OK", false))
                    return false;
              if(f.getFrameType() == 128)
              {
                     haveDIS = true;
                    cap = new ModemCapabilities();
                     byte capb[] = f.getData();
                    for(int i1 = 0; i1 < capb.length; i1++)
                     {
                         int p = capb[i1];
                         if(p < 0)
                         p = 256 + p;
                         capb[i1] = reverseBytes[p];
                     }
                    cap.decodeCapabilities(capb);
              }
              if(f.isLast())
                     break;
              sendATCommandnoWait("AT+FRH=3");
             }
              while(true);
             if(!haveDIS)
             {
              lastError = "DIS not received";
              return false;
          }
          if(!cap.t4)
         {
              lastError = "Receiver does not support T.4";
              return false;
          } else
          {
              return true;
          }
      }
     /*
     readFile - This method is called to read from file.
     Parameters: f - Object of String class.
     Return Value: byte
     */
     protected byte[] readFile(String f)
      {
          byte b[] = null;
       try
       {
           FileInputStream fi = new FileInputStream(f);
           b = new byte[(int)(new File(f)).length()];
           fi.read(b, 0, b.length);
           fi.close();
       }
       catch(Exception e)
       {
          System.err.println(String.valueOf(String.valueOf(
          (new StringBuffer("Read file ")).append(f).append(" ").append(e.getMessage()))));
           byte abyte0[] = null;
           return abyte0;
       }
       return b;
  }
/*
readFile - This method is called to test bit rate.
Parameters: NA
Return Value:boolean
*/
 protected boolean doTraining()
 {
  boolean reTry = true;
  LCFrame f = null;
  while(reTry)
  {
      sendTraining();
      reTry = false;
      sendATCommand("AT+FTS=1");
      sendATCommandnoWait("AT+FRH=3");
      do
```

```
        {
            f = waitForFrame(30000);
            if(f == null)
                return false;
        } while(!f.isLast());
        if(f.getFrameType() == 68)
        {
           lastError = "Training failed at speed ".concat(String.valueOf(String.valueOf(cap.rate)))
               reTry = true;
               int newRate2 = 1;
               if(bitrate == 5)
              newRate2 = 3;
               if(bitrate == 4)
              newRate2 = 3;
               if(bitrate == 3)
              newRate2 = 1;
               if(bitrate == 2)
              newRate2 = 1;
               if(bitrate == 1)
              newRate2 = 0;
               if(bitrate == 0)
              reTry = false;
               if(reTry)
               {
                   bitrate = newRate2;
                   lastError = "";
               }
               sendClass1FTH(3);
               if(!sendDCS())
                   return false;
              }
          }
          if(f != null && f.getFrameType() != 132)
          {
             lastError = "Confirmation to receive excepted.";
             return false;
        }
        return waitForOK("OK", false);
    }
    /*
    sendDataClass1 - This method is called to fragment the data.
    Parameters: b - array of byte type, len - variable of integer type,
    ignoreLineScan - variable of boolean type.
    Return Value:boolean
     */
       protected boolean sendDataClass1(byte b[], boolean ignoreLineScan)
        {
         int count = 0;
         int totalCount = 0;
         int len = b.length;
         byte chunk[] = new byte[buffersize];
         int bytesLine = 0;
         int minBytesLine = 0;
         int lineNumber = 0;
         String logLin = "Line: ";
         boolean error = false;
         if(!ignoreLineScan && cap.scanTime > 0)
         {
             double scanTimeSecs = (double)cap.scanTime / (double)1000;
             minBytesLine = (int)((double)((cap.rate * 100) / 8) * scanTimeSecs);
         }
         if(log)
             addLogLn(String.valueOf(String.valueOf((new StringBuffer
             ("Number of byte/line ")).append(minBytesLine).append
             (" (").append(cap.scanTime).append(" ms )"))));
            if(!sendClass1FTM(cap.rate))
                return false;
            try
            {
                if(flowcontrol == 1)
                serialPort.setFlowControlMode(8);
            }
            catch(Exception exception) { }
            try
            {
                bytesLine = 0;
                do
                {
                   if(totalCount >= b.length || error)
                     break;
                    count = 0;
                    do
                    {
                      if(count >= buffersize || totalCount >= b.length || error)
                      break;
                     byte dataByte = 0;
                     byte nextByte = 0;
                     boolean EOL = false;
```

```
                if(totalCount < b.length - 1)
                nextByte = b[totalCount + 1];
                if((b[totalCount] & 7) == 0 && nextByte == 1)
                EOL = true;
               if(bitorder)
               {
                   int p = b[totalCount++];
                  if(p < 0)
                   p = 256 + p;
                   dataByte = reverseBytes[p];
               }
               else
               {
                   dataByte = b[totalCount++];
               }
                   if(dataByte == 16)
                   chunk[count++] = 16;
                   chunk[count++] = dataByte;
                   bytesLine++;
                   if(EOL && lineNumber > 0 && bytesLine < minBytesLine)
                   for(; bytesLine < minBytesLine; bytesLine++)
                      chunk[count++] = 0;
                   if(EOL)
                   {
                       if(bitorder)
                      chunk[count++] = -128;
                         else
                         chunk[count++] = 1;
                         totalCount++;
                         lineNumber++;
                         bytesLine = 0;
                   }
             } while(true);
             if(!sendBytes(chunk, count))
                 error = true;
            } while(true);
        }
        catch(Exception e)
        {
           System.err.println(e.getMessage());
        }
        try
        {
            if(flowcontrol == 1)
            serialPort.setFlowControlMode(0);
        }
        catch(Exception exception1) { }
        return !error;
    }
   /*
   sendBytes - This method is called to send output bits.
   Parameters: b - array of byte type, len - variable of integer type.
   Return Value:boolean
   */
   protected boolean sendBytes(byte b[], int len)
   {
   try
     {
          addLog(String.valueOf(String.valueOf((new StringBuffer
          ("Send ")).append(len).append(" bytes  ... "))));
          outputStream.write(b, 0, len);
          addLogLn("OK");
     }
     catch(IOException e)
     {
          System.err.println(e.getMessage());
          boolean flag = false;
          return flag;
     }
     return true;
 }
/*
sendData - This method is called to invoke sendBytes method.
Parameters: b - array of byte type
Return Value:boolean
*/
protected boolean sendData(byte b[])
{
   int count = 0;
     int totalCount = 0;
     int len = b.length;
     byte chunk[] = new byte[65];
     try
     {
          while(totalCount < b.length)
          {
             for(count = 0; count < 64 && totalCount < b.length;)
             {
```

```
                byte dataByte = 0;
                if(bitorder)
                {
                   int p = b[totalCount++];
                   if(p < 0)
                   p = 256 + p;
                   dataByte = reverseBytes[p];
                }
                 else
                 {
                   dataByte = b[totalCount++];
                 }
                 if(dataByte == 16)
                 chunk[count++] = 16;
                 chunk[count++] = dataByte;
               }
               if(!sendBytes(chunk, count))
               {
                  boolean flag = false;
                  return flag;
             }
             if(!log);
          }
      }
      catch(Exception e)
      {
          System.err.println(e.getMessage());
      }
      return true;
  }
/*
waitForFrame - This method is called to receive frames.
Parameters: tiout - array of byte type
Return Value: LCFrame
 */
protected LCFrame waitForFrame(int tiout)
 {
      Calendar cal = Calendar.getInstance();
      long startTime = cal.getTime().getTime();
      long startFrameTime = 0L;
      String line = "";
      boolean dle = false;
      boolean started = false;
      byte bytes[] = new byte[1024];
      int byteCount = 0;
      try
      {
          if(log)
             addLog("Wait for frame: ");
          int c = inputStream.read();
       do
       {
          long now = Calendar.getInstance().getTime().getTime();
          if(now - startTime > (long)tiout)
       {
          cancelReception();
          lastresponse = 9;
          lastError = "Timeout waiting for frame";
          LCFrame hdlcframe = null;
          return hdlcframe;
       }
        if(c == 13)
        {
            if(line.indexOf("ERROR") >= 0)
            {
                lastresponse = 1;
                LCFrame hdlcframe1 = null;
                return hdlcframe1;
            }
            if(line.indexOf("FCERROR") >= 0)
            {
                lastresponse = 1;
                LCFrame hdlcframe2 = null;
                return hdlcframe2;
            }
            if(line.indexOf("OK") >= 0)
            {
                lastresponse = 10;
                LCFrame hdlcframe3 = null;
                return hdlcframe3;
            }
            line = "";
        }
        if(log)
       addLog(" ".concat(String.valueOf(String.valueOf(Integer.toHexString(c)))));
        if(c == 255 && !started)
        {
             started = true;
```

```
                    startFrameTime = cal.getTime().getTime();
                }
                if(started)
                if(!dle)
                {
                    if(c == 16)
                    dle = true;
                    else
                    bytes[byteCount++] = (byte)c;
                 }
                 else
                 {
                    if(c == 16)
                    {
                       dle = false;
                       bytes[byteCount++] = (byte)c;
                       }
                       if(c == 3)
                           break;
                   }
                    c = inputStream.read();
                     if(c != 10 && c != 13)
                    line = String.valueOf(line) + String.valueOf((char)c);
                 } while(true);
              }
              catch(Exception e)
              {
              System.err.println(e.getMessage());
              LCFrame hdlcframe4 = null;
              return hdlcframe4;
           }
           if(byteCount == 0)
               return null;
           else
            return new LCFrame(bytes, byteCount);
 }
protected boolean sendClass1Disconnect()
{
   if(!sendClass1FTH(3))
   return false;
   LCFrame DSCN = new LCFrame();
   DSCN.setLast(true);
   DSCN.setFrameType((byte)-5);
   if(!sendFrame(DSCN))
   return false;
   return waitForOK("OK", false);
}
protected boolean sendClass1FET(boolean lastPage)
{
   if(!sendClass1FTH(3, 3))
         return false;
    LCFrame FET = new LCFrame();
    FET.setLast(true);
    if(lastPage)
       FET.setFrameType((byte)47);
    else
         FET.setFrameType((byte)79);
    if(!sendFrame(FET))
         return false;
    return waitForOK("OK", false);
 }
protected boolean sendTSI()
{
 if(!sendClass1FTH(3))
      return false;
    LCFrame TSI = new LCFrame();
    TSI.setLast(false);
    TSI.setFrameType((byte)67);
    for(int i = ownid.length() - 1; i >= 0; i--)
       TSI.addByte((byte)ownid.charAt(i));
    for(int i = 0; i < 20 - ownid.length(); i++)
         TSI.addByte((byte)32);
    if(!sendFrame(TSI))
         return false;
    return waitForOK("CONNECT", false);
 }
 protected boolean sendDCS()
 {
     LCFrame DCS = new LCFrame();
     DCS.setLast(true);
     DCS.setFrameType((byte)-125);
     minCapabilities(cap);
     byte bcap[] = cap.encodeCapabilities();
     for(int i1 = 0; i1 < bcap.length; i1++)
     {
         int p = bcap[i1];
         if(p < 0)
             p = 256 + p;
```

```
            bcap[i1] = reverseBytes[p];
        }
        DCS.addByte(bcap[0]);
        DCS.addByte(bcap[1]);
        DCS.addByte(bcap[2]);
        if(!sendFrame(DCS))
            return false;
        return waitForOK("OK", false);
    }
    /*
    hangup - This method is called to hang-up the modem, killing the dial tone.
    Parameters: NA
    Return Value: NA
    */
    protected void hangup()
        {
          sendATCommand("ATH");
        }
    /*
    sendTraining - This method is called to set flow control of modem and invoke sendBytes method.
    Parameters: NA
    Return Value:boolean
    */
    protected boolean sendTraining()
    {
       /*
       Terminates transmission and waits for 7*10ms interval before responding with OK.
       ERROR is issued if the modem is on-hook.
       */
       sendATCommand("AT+FTS=7");
       if(!sendClass1FTM(cap.rate))
          return false;
            try
            {
                if(flowcontrol == 1)
                serialPort.setFlowControlMode(8);
            }
            catch(Exception exception) { }
            byte b[] = new byte[(int)((double)((cap.rate * 100) / 8) * TRAINING_SECONDS) + 1];
            for(int i = 0; i < b.length; i++)
              b[i] = 0;
          b[b.length - 1] = 1;
          sendBytes(b, b.length);
          byte b1[] = {16, 3};
          sendBytes(b1, b1.length);
          try
          {
              if(flowcontrol == 1)
             serialPort.setFlowControlMode(0);
            }
            catch(Exception exception1) { }
            return waitFor("OK");
        }
       protected boolean sendClass1FTH(int rate)
        {
          return sendClass1FTH(rate, timeout);
        }
       /*
       sendClass1FTM - This method is called to set transmits data using
       HDLC protocol and the defined modulation.
       Parameters: rate - Variable of integer type, pout - variable of integer type.
       Return Value:boolean
       */
       protected boolean sendClass1FTH(int rate, int pout)
       {
          String fth = "AT+FTH=".concat(String.valueOf(String.valueOf(rate)));
          int retry = 0;
          do
          {
             if(retry >= 3)
             break;
             if(!sendATCommandnoWait(fth))
             return false;
             if(waitForOK("CONNECT", false, pout))
             break;
             retry++;
          }
          while(true);
          return retry < 3;
    }
    /*
    sendClass1FTM - This method is called to set transmits data according to the defined modulation
    Parameters: rate - Variable of integer type.
    Return Value:boolean
    */
    protected boolean sendClass1FTM(int rate)
        {
          String ftm = "AT+FTM=".concat(String.valueOf(String.valueOf(rate)));
```

```
          if(!sendATCommandnoWait(ftm))
          return false;
          return waitForOK("CONNECT", false);
        }
        protected void cancelReception()
        {
         if(log)
             addLogLn("Cancel: ");
            byte cancelByte[] = {24};
            sendBytes(cancelByte, 1);
            waitFor("OK");
     }
    /*
    sendFrame - This method is called to send frames.
    Parameters: f - object of LCFrame class.
    Return Value:boolean
    */
    protected boolean sendFrame(LCFrame f)
     {
         byte b[] = f.getRawData();
         if(log)
         {
             addLogLn("");
             addLog("Send frame: ");
             for(int i = 0; i < b.length; i++)
             addLog(" ".concat(String.valueOf(String.valueOf(Integer.toHexString(b[i] & 0xff)))));
          }
          startTimer(2500);
          if(!sendBytes(b, b.length))
                      return false;
                    byte endOfFrame[] = {16, 3};
                    if(!sendBytes(endOfFrame, 2))
                     return false;
                    if(timeout())
                    {
                        lastError = "timeout sending frame.";
                        return false;
                    }
                    else
                    {
                        return true;
                    }
                 }
                /*
                minCapabilities - This method is called to set modem capabilities.
                Parameters: cap - object of ModemCapabilities class.
                Return Value: NA
                */
                protected void minCapabilities(ModemCapabilities cap)
                {
                   int desiredRate = brClass2_to_Class1[bitrate];
                  if(desiredRate < cap.rate)
                  {
                      cap.rate = desiredRate;
                      cap.vRate = brClass2_to_Class1V[bitrate];
                  }
                  if(cap.resolution == 1 && resolution == 0)
                      cap.resolution = 0;
                  if(encoder.lineWidth < cap.width)
                      cap.width = encoder.lineWidth;
                  cap.huffman = 1;
              }
             /*
             startTimer - This method is called to set starting time.
             Parameters: t - Integer type variable.
             Return Value: NA
             */
             protected void startTimer(int t)
             {
               tout = t;
               Calendar cal = Calendar.getInstance();
               startTime = cal.getTime().getTime();
             }
             /*
             timeout - This method is called to check time out.
             Parameters: NA
             Return Value:boolean
             */
             protected boolean timeout()
             {
                long now = Calendar.getInstance().getTime().getTime();
                return now - startTime > tout * (long)1000;
              }
}
```

Download this listing.

In the above listing, the SendingFax class sends a fax to the fax number of the destination. The various methods defined in the listing are:

- `addLogLn()`:dds a line that depicts the status of the Fax application in a log file.
- `addLogCh()`:Adds a character that depicts the status of the modem to the log file.
- `readFile()`:Reads the log file that contains status information about the modem.
- `openForFax()`:Opens the selected port to send the fax.
- `createFaxFiles()`:Creates an image for the file that is open in the edit pane of the Fax Application window.
- `setPortName()`:Sets the name of the port that sends the fax.
- `setInitString()`:Sets the initial string for the modem.
- `writeFile()`:Writes the file that is open in the edit pane of the Fax Application window to a temporary file on a physical drive. The createFaxFiles() method invokes the writeFile() method to write the file.
- `sendATCommand()`:Sends the AT command to the modem. The AT command checks if the modem is ready to send the fax.
- `sendATCommandnoWait()`:Sends the wait command to the modem.
- `softDelay()`: Suspends the processing of the main thread for a specified period to display the status of the Fax application. The end user specifies the duration of the delay.
- `queryCapabilities()`:Retrieves the encoding and decoding capabilities of the modem.
- `waitFor()`:Invokes the waitForOK() method.
- `waitForOK()`:Waits for the OK command that the modem sends if the modem is ready for communication.
- `close()`:Closes the input and output streams of the Fax application.
- `sendFax()`:Sends the fax document to the recipient number specified in the Fax Number text box of the Fax Application window.
- `connect()`:Connects to the modem.
- `sendDataClass1()`:Sends the fax document if the modem used is of Class 1 type. The three different classes of modems are Class 1, Class 2, and Class 2.0.
- `sendBytes()`:Helps the modem send the fax document in the form of bytes to the recipient.
- `sendData()`:Invokes the sendBytes() method to send the fax document.
- `waitForFrame()`:Sets the delay time between two frames.
- `sendClass1Disconnect()`: Disconnects a Class 1 modem.
- `sendClass1FET()`:Sets the number of bytes in a frame.
- `sendTSI()`:Sends the frames of the fax document to the Transmit Station ID (TSI) modem and waits for a response from the modem.
- `sendDCS()`:Determines the type of modem, and sends the frames of the fax document to the modem if the modem type is Digital Command Signal (DCS).
- `hangup()`:Disconnects the modem.
- `sendTraining()`:Sets the flow control property of the modem and invokes the sendBytes() method.
- `sendClass1FTH()`:Transmits data using the High-Level Data Link Control (HDLC) protocol and the defined modulation.
- `sendClass1FTM()`:Transmits data according to the defined modulation.
- `cancelReception()`:Sends the cancel command to the modem.
- `sendFrame()`:Sends the frames of the fax document to the recipient.
- `minCapabilities()`:Retrieves the minimum encoding and decoding capabilities of the modem.
- `startTimer()`:Starts a timer for the Fax application.
- `timeout()`:Closes the connection if the recipient does not respond for a specific period.

Unit Testing

To test the Fax application:

1. Download the Javacomm20-win32 JCA, which is available in a zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where J2SDK is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:
   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```
7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:
   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```
8. Copy the Java files for the Fax application to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:
   ```
   javac *.java
   ```
9. Run the Fax application using the following command at the command prompt:
   ```
   java PortFax
   ```
10. Select File->Open to open the document that you want to send as a fax using the Fax application. The selected file opens in the edit pane of the Fax Application window, as shown in Figure 6-4:

Figure 6-4: The Fax Application Window

11. Select the HTML check box if you want to send the fax document as an HTML page.
12. Specify the fax number in the Fax Number text box.
13. Select File->View Properties to set the properties of the port to which the modem is connected.
14. Click the Send Fax button to send the document as a fax document.
15. Click the Close button to close the Fax application.

Index

A-J

Index

M-U

List of Figures

List of Tables

Chapter 1: Introducing the Java Communication API

List of Listings

Chapter 2: Creating a Port Information Application

Chapter 3: Creating the Modem Dialer Application

Chapter 4: Creating a Printing Application

Chapter 5: Creating a Serial Communication Application

Chapter 6: Creating a Fax Application

Contents

Related Links:

InstantCode Series

Collections:

BusinessPro

ITPro

Java InstantCode: Developing Applications Using JCA

SkillSoft Press © 2004

This book describes how JCA defines a development methodology and patterns that are useful both for building connectors to legacy applica adapters for new applications.

Team LiB

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

Java provides Java Communication Application Programming Interface (JCA), which contains classes and interfaces that help you develop platform-independent communication applications based on technologies, such as voice, mail, fax, or smart cards. JCA provides the javax.comm package, which contains interfaces, classes, and exception classes to interact with the RS232 serial and parallel ports.

This book describes how JCA defines a development methodology and suggests analysis and design patterns that are useful both for building connectors to legacy applications and for designing adapters for new applications.

About the Author

Vibha holds a Bachelor's degree in IT Engineering. She is proficient in technologies such as C++, Java, HTML, DHTML, J2EE, EJB, JNDI, JMS, and RMI. In addition, she has written books on Oracle, jDeveloper, and jBuilder for NIIT.

Credits

I would like to thank Reena Roy, S. Sripriya, Gaurav Bhatla, and Anurag for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

Copyright

Java InstantCode: Developing applications using JCA

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introducing the Java Communication API

Java provides Java Communication Application Programming Interface (JCA), which contains classes and interfaces that help you develop platform-independent communication applications based on technologies, such as voice, mail, fax, or smart cards. JCA provides the javax.comm package, which contains interfaces, classes, and exception classes to interact with the RS232 serial and parallel ports. The serial ports are also called COM ports.

Apart from serial and parallel ports, there exists other type of ports such as Universal Serial Bus (USB) ports. A USB port provides just one standard plug-in for all peripherals in comparison to serial or parallel ports, which provide more than one plug-in for individual devices. A USB port enables you to detect the USB enabled device without restarting your computer.

This chapter explains the RS232 standards for serial ports. It also explains JCA and its classes and interfaces, and describes how to install and set up JCA.

The RS232 Standard

RS232 is a standard that specifies how serial ports communicate. The javax.comm package enables you to develop applications that interact with the devices attached to RS232 serial ports. The RS232 communication standard is based on asynchronous serial communication, which means that communication between the sending device and the receiving device occurs bit by bit. The sending device needs to send the start bit and the stop bit to mark the start and the end of a message.

RS232 Bits Streams

According to the RS232 standard, the port driver breaks the data that needs to be sent into data words. The length of the data words can range from five to eight bits per word. To correctly synchronize the word transfer, the port driver adds some additional bits to the word for error checking. In addition, it is important that the number of bits sent by the sending device must be equal to number of bits received by the receiving device. The additional bits sent with the words are of various types, such as:

- Start bit: Represents the start of a data word. The start bit does not represent the data word that needs to be sent; it is the starting point of the data word. The space line recognizes the start bit.
- Data bit: Represents a bit from the data word. After sending the start bit, the communication application sends the data bits to a data word.
- Parity bit: Represents a bit in the data word for error detection. After sending the data word, the communication application automatically adds a parity bit to the data word. The sending device calculates the value of the parity bit based on the information that it sends. The receiving device also performs the same calculation as the sending device and checks if the actual parity bit value is the same as the calculated value.
- Stop bit: Represents the end of the data word. All data bits and parity bits are within the start and stop bits.

Physical Properties of the RS232 Cable

The RS232 standards define the physical properties of the RS232 cable, such as the voltage level at which the bit will be sent and the maximum length of the cable for communication to take place. The RS232 standards define two states of signal level, high bit and low bit. A negative voltage identifies the high bit state and a positive voltage identifies a low bit state. RS232 also defines standards for the maximum cable length. Per RS232 standards, the maximum cable length is 50 feet. This standard allows communication to occur at the optimum speed.

Error Detection

To detect errors during communication, both the sending and the receiving device calculate the value of the parity bit. If the value of the parity bit after calculation is the same at both ends, the communication is error-free; otherwise, it is not. It is mandatory for both the sender and the receiver to use the same algorithms to check the value of the parity bits sent with the data word. The two types of parity checking are:

- Even parity: In even parity, the number of data bits sent contains an even number of logical 1s. If the number of high data bits is even, the communication application adds a low parity to a communication message; otherwise, the communication application adds a high parity bit.
- Odd parity: The odd parity bit is similar to the even parity bit but in odd parity, the number of high bits is always odd.

Communicating Using the Serial Device

Serial devices communicate using a serial port, which transfers data one bit at a time using a single wire in a data cable. The serial port on a computer supports full-duplex communication. This means that the port can send and receive data at the same time. To send and receive data separately, serial ports use separate wires.

A device attached to a serial port starts communicating by first sending a start bit. The start bit signifies that the data or message is about to start. The device can send five, six, seven, or eight data bits, based on the agreed number of bits, between the device and the serial port. The device and a port must agree on the number of data bits to be sent and received. After data has been sent, the device sends a stop bit to indicate that the transfer of data bits is over. A stop bit has a value of one.

Communicating Using the Parallel Device

Parallel devices communicate using a parallel port, which simultaneously transfers eight bits over eight separate wires in a data cable. The parallel port consists of a connector with 17 signal lines and 8 ground lines. The signal lines include Control, Status, and Data lines.

The Control lines act as an interface control. These lines enable handshaking of signals from the computer to the device attached to a parallel port. The Status lines help in indicating the status of the device attached to the computer, such as printer busy, paper empty, and interface or peripheral errors. The Data lines provide data from the computer to the device attached to a parallel port, such as printer.

PREVIOUS NEXT

Functions of JCA

Using JCA, you can create objects of serial ports and parallel ports. Java applications use the objects to communicate with the ports. The various functions that you can perform using JCA are:

- Open ports for communication.
- Resolve port contention among Java objects.
- Perform asynchronous and synchronous communication with the Java application and the devices attached to the serial and parallel ports.
- Claim ownership or control of a port.
- List the ports available on a computer.
- Send data to and receive data from a port.

The javax.comm Package

JCA provides the javax.comm package, which is a Java extension Application Programming Interface (API). The standard Java Development Kit (JDK) does not provide this API. The javax.comm package provides classes and interfaces that can communicate with serial ports and parallel ports but cannot communicate with other ports, such as USB.

Classes

JCA defines three levels of classes: high-level, low-level, and driver-level classes. High-level classes manage the ownership of and access to communication ports. Low-level classes provide an interface to physical ports. Driver-level classes provide an interface between low-level classes and operation systems.

The javax.comm package contains various classes that you can use to develop applications to interact with the communication ports on a computer. The classes in the javax.comm package are:

- CommPort: Is an abstract class that describes the communication ports available on a particular computer. The CommPort class defines various methods that you can use to control input/output operations on ports. The most commonly used methods in the CommPort class are:
  - `close()`: Closes a communication port.
  - `disableRecieveFraming()`: Is an abstract method that disables receive framing.
  - `disableReceiveTimeout()`: Is an abstract method that disables receive timeout.
  - `enableRecieveFraming()`: Is an abstract method that enables receive framing. The enableRecieveFraming () method is driver-dependent. The method allows receive framing only if driver support enables receive framing.
  - `enableReceiveTimeout()`: Is an abstract method that enables receive time out. The enableReceiveTimeOut() method is driver-dependent.
  - `getInputBufferSize()`: Is an abstract method that returns the size of the input buffer. The getInputBufferSize() returns an integer value.
  - `getOutputBufferSize()`: Is an abstract method that returns the size of the output buffer. The getOutputBufferSize() returns an integer value.
  - `getInputStream()`: Is an abstract method that returns the input stream. The input stream is the only way to receive data from the communication port. The getInputStream() method returns an InputStream type value.
  - `getName()`: Retrieves a string that represents the name of the communication port.
  - `toString()`: Returns the string representation of a communication port.
  - `getOutputStream()`: Returns an output stream. This stream works as a mediator to send data to the communication port.
  - `isReceiveTimeoutEnabled()`: Is an abstract method that verifies whether or not receive timeout is enabled. The isReceiveTimeoutEnabled() method returns a Boolean value.
  - `isReceiveFramingEnabled()`: Is an abstract method that verifies whether or not receive framing is enabled. The isReceiveFramingEnabled() method returns a Boolean value.
  - `setOutputBufferSize()`: Is an abstract method that sets the size of the output buffer.
  - `setInputBufferSize()`: Is an abstract method that sets the size of the input buffer.
- CommPortIdentifier: Controls access and communication between a Java application and ports. The CommPortIdentifier class also defines methods for various operations, such as determining available communication ports, opening communication ports, and determining the ownership of ports. The commonly used methods in the CommPortIdentifier class are:
  - `addPortName()`: Adds the name of a port to the port name list.
  - `addPortOwnershipListener()`: Registers a Java application to receive event notification from the port when a change occurs in the ownership of the port. The three types of events are PORT_OWNED, PORT_UNOWNED, and PORT_OWNERSHIP_REQUESTED. The PORT_OWNED event specifies that a port has an owner. The PORT_UNOWNED event specifies that the port does not have an owner. The PORT_OWNERSHIP_REQUESTED event specifies that a Java application wants to relinquish ownership of the port.
  - `getCurrentOwner()`: Obtains the name of the current owner of a port.
  - `getName()`: Acquires the name of the port.
  - `getPortIdentifier()`: Obtains an enumeration object that contains a list of the available ports. The enumeration object contains objects of type CommPortIdentifier object.

- `getPortType()`: Returns an integer value that represents the port type.
- `open()`: Opens a port for communication, if the port is free. The open() method acquires exclusive ownership of the port if there is no other owner of the requested port.
- `isCurrentlyOwned()`: Checks the ownership of a port, whether it is true or false. If the requested port is owned by a device, the method returns true; and if the requested port is not owned, the method returns false.
- `removePortOwnershipListener()`: Unregisters the CommPortOwnershipListener listener.

- `ParallelPort`: Is a subclass of the CommPort class. The ParallelPort class describes a low-level interface to a parallel port. The most commonly used methods in the ParallelPort class are:
  - `addEventListener()`: Is an abstract method that registers an event listener for a parallel port. You can add only one event listener on one parallel port.
  - `removeEventListener()`: Is an abstract method that unregisters an event listener for a parallel port.
  - `getOutputBufferFree()`: Is an abstract method that retrieves the number of bytes available in the output buffer. The getOutputBufferFree() method returns an integer value.
  - `isPaperOut()`: Is an abstract method that verifies if the port indicates a printer is in the Out of Paper state. The isPaperOut() method returns a Boolean value. For example, the method returns true if the printer's tray does not contain paper; else, the method returns false.
  - `isPrinterBusy()`: Is an abstract method that verifies if the port indicates the Printer Busy state. The isPrinterBusy() method returns a Boolean value. For example, the method returns true if the printer is busy; else, the method returns false.
  - `isPrinterSelected()`: Is an abstract method that verifies if the port indicates a printer in the selected state. The isPrinterSelected() method returns a Boolean value. The method returns true if the printer is in the selected state; else, the method returns false.
  - `isPrinterTimedOut()`: Is an abstract method that verifies if the port indicates that a printer has timed out. The isPrinterTimedOut() method returns a Boolean value. The method returns true if the printer has timed out; else, the method returns false.
  - `notifyOnBuffer()`: Is an abstract method that conveys that the communication application should be notified when the output buffer is empty.
  - `notifyOnError()`: Is an abstract method that conveys that the communication application should be notified when an error occurs on a port.
  - `restart()`: Is an abstract method that resumes output after an error occurs.
  - `getMode()`: Is an abstract method that retrieves the currently configured mode of the port.
  - `setMode()`: Is an abstract method that sets the mode of a printer port. The different printer modes are LPT_MODE_SPP, LPT_MODE_PS2, LPT_MODE_EPP, and LPT_MODE_ECP.
  - `suspend()`: Is an abstract method that suspends output.
  - `removeEventListener()`: Is an abstract method that unregisters an event listener.
  - `isPrinterError()`: Is an abstract method that returns a Boolean value to specify if an error occurred on a printer.

- ParallelPortEvent: Defines the events of the parallel port. The ParallelPortEvent class defines some static variables, such as PAR_EV_ERROR and PAR_EV_BUFFER, which you can use within the methods defined in the class. The PAR_EV_ERROR variable indicates that an error has occurred at a port. The PAR_EV_BUFFER variable indicates that the port output buffer is empty. The methods defined in the ParallelPortEvent class are:
  - `getEventType()`: Returns an integer value that represents the type of event. The getEventType() method may return the PAR_EV_ERROR or PAR_EV_BUFFER static variable.
  - `getNewValue()`: Returns the new value of the changed state of a parallel port. The getNewValue() method returns a Boolean value.
  - `getOldValue()`: Returns the old Boolean value of the state change of a parallel port.

- SerialPort: Is a subclass of CommPort that provides access to a different serial port on a computer. In addition, the SerialPort class also defines methods that allow port to open and close and data to be sent and received. The SerialPort class also defines various static integer variables that the methods defined in the SerialPort class use. The most commonly used methods in the SerialPort class are:
  - `getBaudRate()`: Returns an integer value that represents the current baud rate of a serial port. The getBaudRate() method is an abstract method.

- `getDataBits()`: Returns an integer value that represents the current number of data bits configured on a serial port. The getDataBits() method is an abstract method. The return integer value can be equal to the static variables DATABITS_5, DATABITS_6, DATABITS_7, or DATABITS_8. Table 1-1 describes these variables.
- `getFlowControlMode()`: Returns an integer value that represents the current flow control mode configured on a serial port. The getFlowControlMode() method is an abstract method.
- `getParity()`: Returns an integer value that represents the current parity setting that is configured. The getParity() method is an abstract method.
- `getStopBits()`: Returns an integer value that represents the currently defined stop bits. The getStopBits() method is an abstract method. The returned integer can be equal to the value of the STOPBITS_1, STOPBITS_2, or STOPBITS_3 static variables. Table 1-1 describes the static variables.
- `isCD()`: Is an abstract method that retrieves the state of the Carrier Detect (CR) bit in Universal Asynchronous Receiver/Transmitter (UART). The isCD() method returns a Boolean value.
- `isCTS()`: Is an abstract method that retrieves the state of the Clear To Send (CTS) bit in UART. The isCTS() method returns a Boolean value.
- `isDSR()`: Is an abstract method that retrieves the state of the Data Set Ready (DSR) bit in UART. The isDSR() method returns a Boolean value.
- `isDTR()`: Is an abstract method that retrieves the state of the Data Terminal Ready (DTR) bit in UART. The isDTR() method returns a Boolean value.
- `isRI()`: Is an abstract method that retrieves the state of the Ring Indicator (RI) bit in UART. The isRI() method returns a Boolean value.
- `isRTS()`: Is an abstract method that retrieves the state of the Request To Send (RTS) bit in UART. The isRTS() method returns a Boolean value.
- `notifyOnBreakInterrupt()`: Is an abstract method that conveys that the communication application must be receive notification from the port when there is a break interrupt on the line.
- `notifyOnCarrierDetect()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Carrier Detect (CD) bit changes.
- `notifyOnCTS()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Clear To Send (CTS) bit changes.
- `notifyOnDataAvailable()`: Is an abstract method that conveys that the communication application must receive event notification from the port when input data is available.
- `notifyOnDSR()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the Data Set Ready (DSR) bit changes.
- `notifyOnFramingError()`: Is an abstract method that conveys that the communication application must receive event notification from the port when a framing error occurs.
- `notifyOnOutputEmpty()`: Is an abstract method that conveys that the communication application must receive event empty.
- `notifyOnOverrunError()`: Is an abstract method that conveys that the communication application must receive event notification from the port when an overrun error occurs.
- `notifyOnParityError()`: Is an abstract method that conveys that it must receive event notification from the port when a parity bit error occurs.
- `notifyOnRingIndicator()`: Is an abstract method that conveys that the communication application must receive event notification from the port when the RI bit changes.
- `removeEventListener()`: Is an abstract method that unregisters the registered event.
- `setDTR()`: Is an abstract method that sets the Data Terminal ready (DTR) bit in URAT.
- `setFlowControlMode()`: Is an abstract method that retrieves the current flow control mode.
- `setRTS()`: Is an abstract method that sets the DTR bit in URAT.

The SerialPort class also defines some static variables that the methods defined in the SerialPort class use, as described in Table 1-1:

Table 1-1: Static Variables of the SerialPort Class

Variable Name	Description
DATABITS_5	Represents 5-data bit format.
DATABITS_6	Represents 6-data bit format.
DATABITS_7	Represents 7-data bit format.

DATABITS_8	Represents 8-data bit format.
STOPBITS_1	Represents that the number of STOP bits is 1.
STOPBITS_2	Represents that the number of STOP bits is 2.
STOPBITS_1_5	Represents that the number of STOP bits is 1-1/2.
PARITY_NONE	Represents that there is no parity bit.
PARITY_ODD	Represents an ODD parity scheme.
PARITY_EVEN	Represents an EVEN parity scheme.
PARITY_MARK	Represents a MARK parity scheme.
PARITY_SPACE	Represents a SPACE parity scheme.
FLOWCONTROL_NONE	Represents that flow control is off.
FLOWCONTROL_RTSCTS_IN	Represents RTC/CTS flow control on input.
FLOWCONTROL_RTSCTS_OUT	Represents RTC/CTS flow control on output.
FLOWCONTROL_XONXOFF_IN	Represents XON/XOFF flow control on input.
FLOWCONTROL_XONXOFF_OUT	Represents XON/XOFF flow control on output.

- SerialPortEvent: Defines the events of the serial port. The various methods defined in the SerialPortEvent class are:
    - `getEventType()`: Returns an integer in the form of a static variable that represents the type of event, such as Break Interrupt (BR), Framing Error (FE), Overrun Error (OR), OUTPUT_BUFFER_EMPTY, DATA_AVAILABLE, or Parity Error (PE). For example, the DATA_AVAILABLE variable indicates that data is available on the serial port.
    - `getNewValue()`: Returns a new value for the state change of a serial port. The getNewValue() method returns a Boolean value.
    - `getOldValue()`: Returns the old value of the state change of a serial port. The getOldValue() method returns a Boolean value.

Interfaces

The javax.comm package provides interfaces that you can use to develop applications. You use the interfaces to communicate with the communication ports. The various interfaces defined in the javax.comm package are:

- CommDriver
- CommPortOwnershipListener
- ParallelPortEventListener
- SerialPortEventListener

The CommDriver interface is a part of a loadable device driver interface. The methods defined in the CommDriver interface are:

- `initialize()`: Registers the port name with the CommPortIdentifier class. In addition, the initialize() method also loads any required native libraries. The initialize() method is an abstract method.
- `getCommPort()`: Returns an object that extends the SerialPort or ParallelPort class. The open() method of the CommPortIdentifier class invokes the getCommPort() method.

The CommPortOwnershipListener interface broadcasts various port ownership events across an application. Opening a port triggers and broadcasts the CommPortOwnershipListener event of type PORT_OWNED. Closing a port triggers and broadcasts the CommPortOwnershipListener event of type PORT_UNOWNED.

The ParallelPortEventListener interface broadcasts the events of parallel ports using the parallelEvent() method. The parallelEvent() is the only method defined in the ParallelPortEventListener interface.

The SerialPortEventListener interface broadcasts the events of serial ports using the serialEvent() method. The serialEvent() is the only method defined in the SerialPortEventListener interface.

Exception Handling

The javax.comm package provides three exception classes that you can use to develop communication applications. An exception class catches the different exceptions that a communication application throws when you use the javax.comm package. Table 1-2 describes various exception classes:

Table 1-2: Exception Classes

Exceptions Classes	Description
NoSuchPortException	Catches the exception that the Java application throws when the application is unable to find the specified port.

PortInUseException	Catches the exception that the Java application throws if the specified port is in use.
UnsupportedCommOperationException	Catches the exception that the Java application throws when the port driver does not allow you to perform a specified operation on the port.

PREVIOUS NEXT

How to Install JCA

To use the features that JCA provides, you need to configure the javax.comm package and the setup environment to execute your communication applications. To install JCA and configure the package setting:

1. Download the Javacomm20-win32 JCA, which is available in zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip javacomm20-win32.zip in the C drive.

3. Copy win32comm.dll to the %JAVA_HOME%\bin directory. %JAVA_HOME% represents the directory that stores JDK on the computer. To copy win32comm.dll, specify the following command at the command prompt:

   ```
   C:\>copy c:\commapi\win32com.dll %JAVA_HOME%\bin
   ```

4. Copy the comm.jar file to the %JAVA_HOME%\lib directory by specifying the following command at the command prompt:

   ```
   C:\>copy c:\commapi\comm.jar %JAVA_HOME%\lib
   ```

5. Copy javax.comm.properties to the %JAVA_HOME%\lib directory by specifying the following command at the command prompt:

   ```
   C:\>copy c:\commapi\javax.comm.properties %JAVA_HOME%\lib
   ```

6. Add the comm.jar file to your classpath by specifying the following command at the command prompt:

   ```
   set classpath = %classpath%;%JAVA_HOME\lib; %JAVA_HOME\jre\lib\ext\comm.jar;
   ```

Chapter 2: Creating a Port Information Application

The Java Communication API (JCA) supports the javax.comm package, which provides the CommPortIdentifier class to control access to communications ports. For example, you can determine the ports attached to a computer, open any specific port for Input/Output (I/O) operations, view the description of the port, and determine the ownership of the port by using methods in the CommPortIdentifier class.

This chapter explains how to develop a Port Information application, which uses the javax.comm package to list all the serial and parallel ports attached to a computer and provide information on ports selected by an end user. It also explains how to open the ports for Input/Output (I/O) operations.

Architecture of the Port Information Application

The Port Information application uses the following files:

- PortDescription.java: Creates a user interface that an end user can use to list the parallel and serial ports, open a selected port, and view the description of the selected port.
- ShowDescriptionWindow.java: Displays the tabular description of the selected serial or parallel port.

Figure 2-1 shows the architecture of the Port Information application:

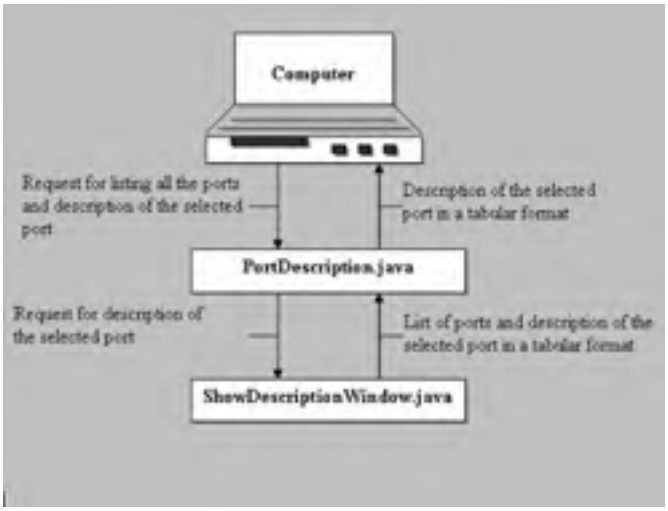

Figure 2-1: Architecture of the Port Information Application

The PortDescription.java file calls the ShowDescriptionWindow.java file, when the end user selects a serial or parallel port from the populated list of ports and clicks the Port Description button.

The ShowDescriptionWindow.java file displays the description of the selected port. The ShowDescriptionWindow.java file displays the various fields related to the selected port which includes Port Type, Port Name, Port Mode, Port Status, and Owner Name to the end user.

Creating the User Interface

The PortDescription.java file creates a user interface, which lets the end user list all serial and parallel ports attached to a computer and open any of the selected port for input/output operations using the CommPortIdentifier class. The PortDescription.java file uses the CommPort and ParallelPort classes to enable the end user to communicate through the selected port opened by the CommPortIdentifier class.

Listing 2-1 shows the code of the PortDescription.java file:

Listing 2-1: The PortDescription.java File

```
/* Imports required Comm classes. */
import javax.comm.*;
/* Imports required I/O classes. */
import java.io.*;
/* Imports required AWT classes. */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
/* Imports required Util classes. */
import java.util.*;
/* Class PortDescription - This class is the main class of the application.
This class initializes the interface and loads all components like the button and
listbox before displaying the result.
Constructor:
PortDescription - This constructor creates GUI.
Methods:
getPortDescription - This method gives port description.
openPort - This method opens a selected port.
getSerialPorts - This method fills list of serial ports.
getParallelPorts - This method fills list of parallel ports.
emptyList - This method clears a list.
main - This method creates the main window of the application and displays all the components.
*/
public class PortDescription implements ActionListener,ListSelectionListener
{
   /* Declare object of JFrame class. */
   JFrame frame;
   /* Declare object of Enumeration class. */
   Enumeration pList;
   /* Declare objects of JLabel class. */
   private JLabel apppagetitle;
   private JLabel seriallabel;
   private JLabel parallallabel;
   /* Declare objects of JButton class. */
   private JButton serialportbutton;
   private JButton commportbutton;
   private JButton openportbutton;
   private JButton portdescriptionbutton;
   /* Declare objects of JList class. */
   private JList listserial;
   private JList listcomm;
   /*
   Declare objects of DefaultListModel class.
   */
   private DefaultListModel listModelserial;
   private DefaultListModel listModelcomm;
   int descriptionpos;
   int buttonwidth=140;
   int buttonheight=25;
   public PortDescription()
   {
      /*
      Initialize and set the look and feel of the application to Windows Look and Feel.
      */
       try
      {
          UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         /*
         If an error occurs while loading the Windows Look and Feel, an
         error is printed and the application is closed.
         */
         System.out.println("Error setting window environment: " + e);
      }
      try
      {
         /*
         Initialize the object of the JFrame class and set the Title.
         */
         frame=new JFrame("Ports Application");
```

```
/*
Initialize the object of the JButton class and set the Title.
*/
Container pane=frame.getContentPane();
/*
Set the layout of the frame as Null.
*/
pane.setLayout(null);
/*
Set background color as white.
*/
pane.setBackground(Color.white);
/*
Initialize a new button, Set hot key as 'S' and add to content pane.
*/
serialportbutton=new JButton("List Serial Ports");
serialportbutton.setMnemonic('S');
serialportbutton.setActionCommand("serial");
serialportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
serialportbutton.addActionListener(this);
pane.add(serialportbutton);
/*
Initialize a new button, Set hot key as 'C' and add to content pane.
*/
commportbutton=new JButton("List Parallel Ports");
commportbutton.setMnemonic('C');
commportbutton.setActionCommand("parallel");
commportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
commportbutton.addActionListener(this);
pane.add(commportbutton);
/*
Initialize a new button, Set hot key as 'O' and add to content pane.
*/
openportbutton=new JButton("Open Port");
openportbutton.setMnemonic('O')
openportbutton.setActionCommand("open");
openportbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
openportbutton.addActionListener(this);
pane.add(openportbutton);
/*
Initialize a new button, Set hot key as 'D' and add to content pane.
*/
portdescriptionbutton=new JButton("Port Description");
portdescriptionbutton.setMnemonic('D');
portdescriptionbutton.setActionCommand("description");
portdescriptionbutton.setPreferredSize(new Dimension(buttonwidth, buttonheight));
portdescriptionbutton.addActionListener(this);
pane.add(portdescriptionbutton);
openportbutton.setEnabled(false);
/* Initialize a new label and add to content pane. */
seriallabel=new JLabel("Serial Ports");
seriallabel.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(seriallabel);
/* Initialize a new label and add to content pane. */
parallallabel=new JLabel("Parallel Ports");
parallallabel.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(parallallabel);
/*
Initialize a new label and add to content pane.
*/
apppagetitle=new JLabel("Port Information Application");
apppagetitle.setFont(new Font("Verdana", Font.BOLD, 14));
pane.add(apppagetitle);
/*
Declare and initialize the object of Insets class.
*/
Insets insets = pane.getInsets();
/*
Initialize the object of DefaultListModel class.
*/
listModelserial = new DefaultListModel();
/*
Initialize the object of DefaultListModel class.
*/
listModelcomm = new DefaultListModel();
/*
Initialize the object of JList class.
*/
listserial = new JList(listModelserial);
listcomm=new JList(listModelcomm);
/*
Set the selection mode of listserial as single selection.
Declare and initialize the object of JScrollPane. Set dimension.
Add list to scroll pane and scroll pane to frame's content pane.
*/
listserial.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
listserial.addListSelectionListener(this);
listserial.setVisibleRowCount(10);
```

```
   JScrollPane listScrollPane = new JScrollPane(listserial);
   listScrollPane.setPreferredSize(new Dimension(250,200));
   Dimension size=listScrollPane .getPreferredSize();
   listScrollPane.setBounds(80, 90, size.width, size.height);
   pane.add(listScrollPane);
   /*
   Set the selection mode of listcomm as single selection.
   Declare and initialize the object of JScrollPane.
   Set dimension. Add list to scroll pane add scroll pane to frame's content pane.
   */
   listcomm.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   listcomm.addListSelectionListener(this);
   listcomm.setVisibleRowCount(10);
   listScrollPane = new JScrollPane(listcomm);
   listScrollPane.setPreferredSize(new Dimension(250,200));
   size=listScrollPane .getPreferredSize();
   listScrollPane.setBounds(380, 90, size.width, size.height);
   pane.add(listScrollPane);
   /*
   Declare object of Dimension class and initialize it with getScreenSize() method.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   /*
   Set location at which window will be displayed.
   */
   frame.setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   /* Initialize left margin. */
   descriptionpos=70;
   size = serialportbutton.getPreferredSize();
   /*
   Set position of the serialportbutton button.
   */
   serialportbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height);
   descriptionpos=descriptionpos+size.width;
   size = commportbutton.getPreferredSize();
   /*
   Set position of the commportbutton button.
   */
   commportbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height);
   descriptionpos=descriptionpos+size.width;
   size = portdescriptionbutton.getPreferredSize();
   /*
   Set position of the portdescriptionbutton button.
   */
   portdescriptionbutton.setBounds(descriptionpos+5, 500 + insets.top, size.width, size.height
   descriptionpos=descriptionpos+size.width;
   size = openportbutton.getPreferredSize();
   /*
   Set position of the openportbutton button.
   */
   openportbutton.setBounds(descriptionpos+ 5, 500 + insets.top, size.width, size.height);
   size = seriallabel.getPreferredSize();
   /*
   Set position of the seriallabel label.
   */
   seriallabel.setBounds(80 + insets.left, 60 + insets.top, size.width, size.height);
   size = parallallabel.getPreferredSize();
   /*
   Set position of the parallallabel label.
   */
   parallallabel.setBounds(380 + insets.left, 60 + insets.top, size.width, size.height);
   size=apppagetitle.getPreferredSize();
   /*
   Set position of the apppagetitle label.
   */
   apppagetitle.setBounds(250 + insets.left,  insets.top, size.width, size.height);
   /*
   Set the size of the Application frame.
   */
   frame.setSize(700,600);
   frame.setVisible(true);
   /*
   addWindowListener - It contains the windowClosing() method.
   windowClosing: It is called when the user clicks the cancel button of the Window. It closes
   Parameter:
   we - Object of WindowEvent class.
   Return Value: NA
   */
   frame.addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent we)
      {
         System.exit(0);
      }
   });
}
catch(Exception e)
{
```

```
            System.err.println(e);
            e.printStackTrace();
        }
    }
    /*
    valueChanged: It is called when the user selects a listitem from the list.
    Parameter:
    le - Object of ListSelectionEvent class.
    Return Value: NA
    */
    public void valueChanged(ListSelectionEvent le)
    {
        if(le.getSource()==listcomm)
        {
            if (listserial.getSelectedIndex()!=-1)
            {
                if (le.getValueIsAdjusting())
                {
                    listserial.clearSelection();
                }
            }
        }
        else if(le.getSource()==listserial)
        {
            if(listcomm.getSelectedIndex()!=-1)
            {
                if (le.getValueIsAdjusting())
                {
                    listcomm.clearSelection();
                }
            }
        }
    }
    /*
    actionPerformed - This method is called when the user clicks the Get Serial Ports,
    Get Comm Ports, Open Port or Port Description button.
    Parameters:
    ae - An ActionEvent object containing details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent e)
    {
        /*
        This is executed when the end user clicks the Get Serial Ports button.
        */
        if ("serial".equals(e.getActionCommand()))
        {
            getSerialPorts();
        }
        /*
        This is executed when the end user clicks the Get Parallel Ports button.
        */
        else if("parallel".equals(e.getActionCommand()))
        {
            getParallelPorts();
        }
        /*
        This is executed when the end user clicks the Port Description button.
        */
        else if("description".equals(e.getActionCommand()))
        {
            getPortDescription();
        }
        /*
        This is executed when the end user clicks the Open Port button.
        */
        else if("open".equals(e.getActionCommand()))
        {
            openPort();
        }
    }
    /*
    getPortDescription - This method open a new window to show port description of selected port.
    Parameters: NA
    Return Value: NA
    */
    public void getPortDescription()
    {
        /*
        Declare and initialize String objects.
        */
        String name="port";
        String isportopen="Not Open";
        /*
        Declare and initialize CommPort objects.
        */
        CommPort port =null;
        ShowDescriptionWindow descriptionwindow;
```

```
   descriptionwindow=new ShowDescriptionWindow();
   /*
   If the end user selected a port from serial port list
   */
   if (listserial.getSelectedIndex()!=-1)
   {
      name= listModelserial.getElementAt(listserial.getSelectedIndex()).toString();
      descriptionwindow.setPortType("Serial port ");
   }
   /*
   Else, the user selected a port from Parallel port list
   */
   else if(listcomm.getSelectedIndex()!=-1)
   {
      name = listModelcomm.getElementAt(listcomm.getSelectedIndex()).toString();
      descriptionwindow.setPortType("Parallel port ");
   }
   try
   {
      CommPortIdentifier commport = javax.comm.CommPortIdentifier.getPortIdentifier(name);
      try
      {
         /* Try to open port. */
         port= commport.open("PortDescription",20);
         descriptionwindow.isPortOpen("Not Open");
         if(commport.getPortType() == CommPortIdentifier.PORT_PARALLEL)
         {
            ParallelPort pport = (ParallelPort)port;
            int mode = pport.getMode();
            switch (mode)
            {
               case ParallelPort.LPT_MODE_ECP:
               descriptionwindow.setPortMode("ECP");
               break;
               case ParallelPort.LPT_MODE_EPP:
               descriptionwindow.setPortMode("EPP");
               break;
               case ParallelPort.LPT_MODE_NIBBLE:
               descriptionwindow.setPortMode("Nibble Mode");
               break;
               case ParallelPort.LPT_MODE_PS2:
               descriptionwindow.setPortMode("Byte mode");
               break;
               case ParallelPort.LPT_MODE_SPP:
               descriptionwindow.setPortMode("Compatibility mode");
               break;
               default:
            }
            if (port!=null)
            {
               port.close();
            }
         }
      }
      catch(PortInUseException pe)
      {
         descriptionwindow.isPortOpen("Open");
         /*
         Declare the String object and initialize it with a value returned by the getCurrentOwner
         */
         String ownername = commport.getCurrentOwner();
         if( ownername == null)
         {
            descriptionwindow.ownerName("unidenified");
         }
         else
         {
            descriptionwindow.ownerName(ownername);
         }
      }
      descriptionwindow.portName(commport.getName());
      descriptionwindow.showMessage(frame);
      descriptionwindow.setResizable(false);
   }
   catch(NoSuchPortException e)
   {
      descriptionwindow.portNotExists("yes");
   }
}
/*
openPort - This method try to open selected port for communication, if port is in
use or does not exists then this displays an error message to the end user.
Parameters: NA
Return Value: NA
*/
public void openPort()
{
   /* Declare and initialize String objects. */
```

```
      String name="";
      String porttype="Unknown port ";
      CommPort port =null;
      if (listserial.getSelectedIndex()!=-1)
      {
         name= listModelserial.getElementAt(listserial.getSelectedIndex()).toString();
         porttype="Serial port ";
      }
      else if(listcomm.getSelectedIndex()!=-1)
      {
         name = listModelcomm.getElementAt(listcomm.getSelectedIndex()).toString();
         porttype="Parallel port ";
      }
      try
      {
         CommPortIdentifier commport = javax.comm.CommPortIdentifier.getPortIdentifier(name);
         try
         {
            port= commport.open("PortListOpen",20);
            int userchoice=JOptionPane.showConfirmDialog(null,porttype+name+" has been
            successful opened. Click Yes to close this port.","Open Port",JOptionPane.YES_NO_OPTION)
            if (userchoice==0)
            {
               if (port!=null)
               {
                  port.close();
               }
            }
         }
         catch(PortInUseException pe)
         {
            String ownername = commport.getCurrentOwner();
            if( ownername == null)
            {
               JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+".
               This is owned by unidentified application","Open Port",JOptionPane.PLAIN_MESSAGE);
            }
            else
            {
               if ("Port currently not owned".equals(ownername))
               {
                  JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+". "
                  +ownername+".","Open Port",JOptionPane.PLAIN_MESSAGE);
               }
               else
               {
                  JOptionPane.showMessageDialog(null,"Open failed for "+porttype+name+". Port owned
                  +ownername,"Open Port",JOptionPane.PLAIN_MESSAGE);
               }
            }
         }
      }
      catch(NoSuchPortException e)
      {
         JOptionPane.showConfirmDialog(null,e,"Open Port",JOptionPane.YES_NO_OPTION);
      }
   }
   /*
   getSerialPorts - This method uses CommPortIdentifier API to get serial port names.
   Parameters: NA
   Return Value: NA
   */
   public void getSerialPorts()
   {
      emptyList("serial");
      pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
         CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
         if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
            /* Fill list from Serial ports. */
            listModelserial.addElement(cpi.getName());
         }
      }
      enableDisableOpenButton();
   }
   /*
   getParallelPorts - This method uses CommPortIdentifier API to get parallel port names.
   Parameters: NA
   Return Value: NA
   */
   public void getParallelPorts()
   {
      emptyList("parallel");
      pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
```

```
            CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
            if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
            {
               /* Fill list from Serial ports. */
               listModelcomm.addElement(cpi.getName());
            }
         }
         enableDisableOpenButton();
      }
      /*
      emptyList - This method clears all list items from the list box.
      Parameters: listtype - object of the String type.
      Return Value: NA
      */
      public void emptyList(String listtype)
      {
         if (listtype=="serial")
         {
            listModelserial.clear();
         }
         else if(listtype=="parallel")
         {
            listModelcomm.clear();
         }
      }
      /*
      enableDisableOpenButton - This method disables/enables open port button
      if serial and/or parallel port lists are empty/non-empty.
      Parameters: NA
      Return Value: NA
      */
      public void enableDisableOpenButton()
      {
         if ((listModelcomm.getSize()>0) || (listModelserial.getSize()>0))
         {
            openportbutton.setEnabled(true);
         }
         else
         {
            openportbutton.setEnabled(false);
         }
      }
      /*
      Main method that creates the instance of the PortDescription class.
      */
      public static void main(String[] args)
      {
         PortDescription pd=new PortDescription();
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortDescription class. This class generates the main window of the Port Information application, as shown in Figure 2-2:

Figure 2-2: The Port Information Application User Interface

When an end user clicks any button on the Ports Application window, the Port Information application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate method, based on the button that the end user clicks.

For example, when the end user clicks the List Serial Ports button, the actionPerformed() method invokes the getSerialPorts() method. The getSerialPorts() method calls the getPortIdentifiers() method of the CommPortIdentifier class to get the enumeration of ports attached to a computer. Next, the getSerialPorts() method calls the getPortType() method of the CommPortIdentifier class to check for the serial port from the enumeration of ports. If the getPortType() method is equal to the CommPortIdentifier.PORT_SERIAL variable, the port is assumed to be a serial port. The method then identifies all the serial ports attached to a computer and lists the ports in the Serial Ports list box.

The List Parallel Ports button functions in a similar manner as that of the List Serial Ports button.

When an end user clicks the Open Port button, the actionPerformed() method invokes the openPort() method. The openPort() method calls the open() method of the CommPortIdentifier class to open the selected port for various input/output operations. If the selected port is already in use, the Port Information application throws an exception.

When the end user clicks the Port Description button, the actionPerformed() method invokes the getPortDescription() method. The getPortDescription() method retrieves the information of the selected port using the CommPortIdentifier class and creates an object of the ShowDescriptionWindow.java class to display the following information about the selected port:

- Port Type
- Port Name
- Port Mode
- Port Status
- Owner Name

◄ PREVIOUS NEXT ►

Displaying the Port Description

The ShowDescriptionWindow.java file displays the name, type, mode, status, and owner of the selected port get from the PortDescription.java file. The ShowDescriptionWindow.java file displays the description of the selected port in a tabular format.

Listing 2-2 shows the contents of the ShowDescriptionWindow.java file:

Listing 2-2: The ShowDescriptionWindow.java File

```
/* Imports required AWT classes. */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
/*
class ShowDescriptionWindow -
This class is the sub class of the application.
This class displays the table of port description.
*/
class ShowDescriptionWindow extends JFrame
{
   /* Declare the String objects. */
   private String porttype;
   private String portstatus;
   private String ownername;
   private String portname;
   private String portexists;
   private String pmode;
   Border tableborder;
   /* Declare the JButton object. */
   JButton okbutton;
   /* Declare the JTable object. */
   JTable table;
   /*
   Declare the default constructor of the FileList class.
   */
   public ShowDescriptionWindow()
   {
      /* Set title of parent frame. */
      super("Port Description");
      porttype="Unknown port";
      portstatus="Not Open";
      ownername="";
      portname="";
      portexists="No";
      pmode="";
   }
   /*
   setPortType - This method sets the value of porttype field.
   Parameters: ptype - a String object containing port type.
   Return Value: NA
   */
   public void setPortType(String ptype)
   {
      porttype=ptype;
   }
   /*
   isPortOpen - This method sets the value of portstatus field.
   Parameters: openornot - a String object containing YES or NO.
   Return Value: NA
   */
   public void isPortOpen(String openornot)
   {
      portstatus=openornot;
   }
   /*
   ownerName - This method sets the value of ownername field.
   Parameters: openornot - a String object containing owner name.
   Return Value: NA
   */
   public void ownerName(String oname)
   {
      ownername=oname;
   }
   /*
   portName - This method sets the value of portname field.
   Parameters: pname - a String object containing port name.
   Return Value: NA
   */
   public void portName(String pname)
   {
      portname=pname;
```

```
}
public void setPortMode(String portmode)
{
   pmode=portmode;
}
/*
portNotExists- This method sets the value of portexists field.
Parameters: exists - A String object containing information of port validity.
Return Value: NA
*/
public void portNotExists(String exists)
{
   portexists=exists;
}
/*
enableDisableOpenButton - This method opens a window
which contains a port description table and ok button.
Parameters:
pd - a JFrame object.
Return Value: NA
*/
public void showMessage(JFrame pd)
{
   /*
   Initialize and set the look and feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error setting window LAF: " + e);
   }
   /* Get the content pane of this class. */
   Container messagewindowpane=getContentPane();
   /*
   Set the layout of the frame as Null.
   */
   messagewindowpane.setLayout(null);
   /*
   Set the background color as white.
   */
   messagewindowpane.setBackground(Color.white);
   /*
   Initialize a new button and add to content pane.
   */
   okbutton=new JButton("OK");
   messagewindowpane.add(okbutton);
   okbutton.setActionCommand("ok");
   okbutton.addActionListener(new ActionListener()
   {
      /*
      actionPerformed - This method is called when the user clicks the Ok button.
      Parameters:
      ae - an ActionEvent object containing details of the event.
      Return Value: NA
      */
      public void actionPerformed(ActionEvent e)
      {
          ShowDescriptionWindow.this.dispose();
      }
   });
   addWindowListener(new WindowAdapter()
   {
      /*
      windowClosing - This method is called when the user clicks the X button on the top bar.
      Parameters:
      e - an WindowEvent object containing details of the event.
      Return Value: NA
      */
      public void windowClosing(WindowEvent e)
      {
          ShowDescriptionWindow.this.dispose();
      }
   });
   /*
   Declare a Point object and initialize it with the getLocation() method
   */
   Point p=pd.getLocation();
   /*
   Set default location for this frame.
   */
   setLocation((int)p.getX(), (int)p.getY());
   /*
   Declare Dimension object and initialize it with the getPreferredSize() method.
   */
   Dimension size=okbutton.getPreferredSize();
```

```
        okbutton.setBounds(140, 180, size.width, size.height);
        /*
        Declare and initialize the Object array.
        */
        Object[][] rowdata={{" Port Type",porttype},{" Port Name",portname},
        {" Port Mode",pmode},{" Port Status",portstatus},{" Owner Name",ownername}};
        /*
        Declare and initialize the String array.
        */
        String[] columnname={"Text","Description"};
        table=new JTable(rowdata,columnname);
        messagewindowpane.add(table);
        size=table.getPreferredSize();
        tableborder=new EtchedBorder(EtchedBorder.RAISED);
        table.setBorder(tableborder);
        table.setBounds(50, 28, 250, size.height);
        setSize(350,250);
        setVisible(true);
    }
}
```

Download this listing.

The above listing displays the description of the selected port. The ShowDescriptionWindow.java file creates a table that displays the description of the selected port.

◄ PREVIOUS NEXT ►

Unit Testing

To test the Port Information application:

1. Download the JCA, which is available in a zip format with the name Javacomm20-win32.zip. The JCA can be downloaded from the following URL: http://java.sun.com/products/javacomm/downloads/index.html
2. Unzip the zip file downloaded from the above URL.
3. Copy the win32com.dll from the unzipped file to jre\bin directory where the J2SDK is installed.
4. Copy the comm.jar from the unzipped file to the jre\lib\ext directory where the J2SDK is installed.
5. Copy the javax.comm.properties from the unzipped file to the jre\lib directory where the J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:
   ```
   set path = %path%;D:\j2sdk1.4.0_02\bin;
   ```
7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:
   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```
8. Copy the PortDescription.java and ShowDescriptionWindow.java files to a folder on your computer. On the command prompt, use the cd command to move to that folder where you have copied the Java files. Next, compile the files using the javac command as follows:
   ```
   javac *.java
   ```
9. To run the Port Information application, specify the following command at the command prompt:
   ```
   java PortDescription
   ```
10. Click the List Serial Ports button on the Port Information application window to list all the serial ports attached to a computer. The Serial Ports list box displays the serial ports, as shown in Figure 2-3:

Figure 2-3: Displaying the List of Serial Ports

11. Click the List Parallel Ports button to list the parallel ports attached to a computer. The Parallel Ports list box displays a list of parallel ports, as shown in Figure 2-4:

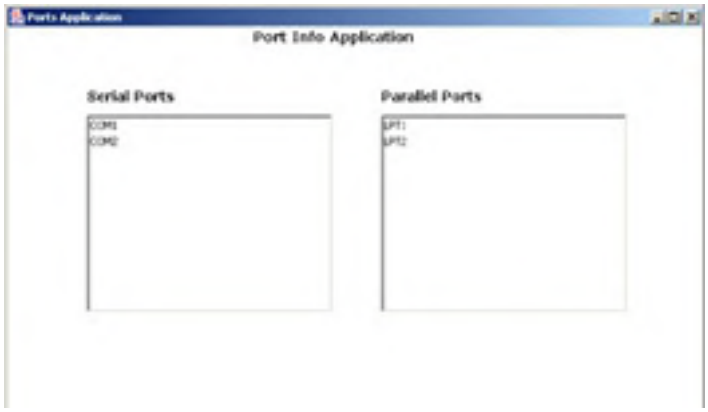

Figure 2-4: Displaying the List of Parallel Ports

12. Select a port from the list of ports and click the Port Description button. The description of the selected port appears, as shown in Figure 2-5:

Figure 2-5: Displaying the Port Description

13. Click the OK button to close the Port Description window.
14. Select a port from the list of ports and click the Open Port button. The selected port opens and a confirmation message appears, as shown in Figure 2-6:

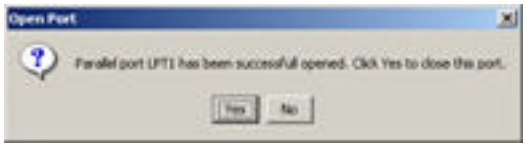

Figure 2-6: The Open Port Window

15. Click the Yes button to close the port. If you clicked the No button and again try to open the same port. The error message appears, as shown in Figure 2-7:

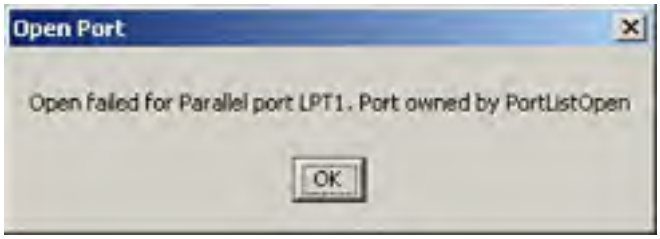

Figure 2-7: An Error Message

PREVIOUS NEXT

Chapter 3: Creating the Modem Dialer Application

The Java Communication Application Programming Interface (JCA) supports the javax.comm package. This package provides the CommPort and ParallelPort classes to obtain a list of the ports on a computer. In addition, the javax.comm package enables you to use the DataInputStream and PrintStream classes of the java.io package to send data to and receive data from the Serial port.

This chapter explains how to develop a Modem Dialer application that uses javax.comm to communicate with a modem attached to a computer and make a phone call to a specified number.

Architecture of the Modem Dialer Application

The Modem Dialer application lists all the serial and parallel ports of a computer. The application also allows an end user to select a port to which the modem is connected and make a phone call to a specified number using the modem.

> **Note** The Modem Dialer application created in this chapter uses the Serial port for the modem.

The Modem Dialer application uses the following files:

- PortModem.java: Creates the user interface of the Modem Dialer application. The end user can use this user interface to specify a phone number.
- PortChooser.java: Creates a user interface in which an end user can select the port to which the modem is connected.

Figure 3-1 shows the architecture of the Modem Dialer application:

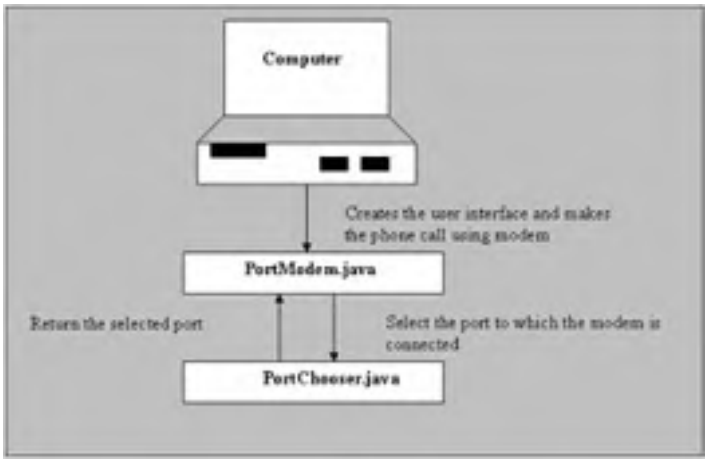

Figure 3-1: The Architecture of the Modem Dialer Application

The PortModem.java file invokes the PortChooser.java file when the end user selects the Browse button to select the port to which the modem is connected from the user interface of the Modem Dialer application.

Creating the User Interface

The PortModem.java file creates the user interface of the Modem Dialer application. Listing 3-1 shows the content of the PortModem.java file:

Listing 3-1: The PortModem.java File

```
/*Imports required Comm package classes.*/
import javax.comm.*;
/*Imports required I/O package classes.*/
import java.io.*;
/*Imports required java.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required AWT classes.*/
import java.awt.*;
import java.awt.event.*;
/*
class PortModem - This class is the main class of the application.
This class initializes the interface and loads all components like the button,
textfields before displaying the result.
Constructor:
   PortModem-This constructor creates GUI.
Methods:
   expect - Get response from modem
   send - send commands to modem
main - This method creates the main window of the application and displays all the components.
*/
public class PortModem extends JFrame implements ActionListener
{
   /*Declare objects of JButton class.*/
   private JButton dialbutton;
   private JButton cancelbutton;
   private JButton closebutton;
   private JButton portbrowsebutton;
   /*Declare objects of JPanel class.*/
   private JPanel bottompanellower;
   private JPanel bottompanelupper;
   private JPanel middlepanel;
   private JPanel bottompanel;
   /*Declare objects of JLabel class.*/
   private JLabel applicationtitlelabel;
   private JLabel portnamelabel;
   private JLabel phonenumberlabel;
   private JLabel processlabel;
   /*Declare objects of JTextField class.*/
   private JTextField portnametextbox;
   private JTextField phonenumbertextbox;
   /*Declare constraints.*/
   private final int BUTTON_WIDTH=80;
   private final int BUTTON_HEIGHT=23;
   private final int TEXTBOX_WIDTH=150;
   public static final int BAUD = 9600;
   protected boolean start = true;
   /*Declare object of DataInputStream class.*/
   protected DataInputStream is;
   /*Declare object of PrintStream class.*/
   protected PrintStream os;
   String response;
   /*
   Declare and initialize object of CommPortIdentifier class.
   */
   CommPortIdentifier commportid=null;
   /*
   Declare and initialize object of CommPort class.
   */
   CommPort commport=null;
   /*Declare object of PortChooser class.*/
   PortChooser portchooser;
   public PortModem() throws IOException, NoSuchPortException,PortInUseException,UnsupportedCommOper
   {
      /*Set parent window title.*/
      super("Modem Dialer Application");
      /*
      Initialize and set the Look and Feel of the application to Windows Look and Feel.
      */
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         System.out.println("Unable to set WLF"+e);
      }
```

```
/*Override windowClosing method.*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
      System.exit(0);
   }
});
/*
Declare and initialize object of Container class.
*/
Container contentpane=getContentPane();
/*Set background color to white.*/
contentpane.setBackground(Color.white);
/*
Declare and initialize object of Dimension class.
*/
Dimension  screendimension=Toolkit.getDefaultToolkit().getScreenSize();
/*Set window's location on screen.*/
setLocation(screendimension.width/2- 200,screendimension.height/2-200);
/*Initialize object of JLabel.*/
applicationtitlelabel=new JLabel("Phone Call Application",JLabel.CENTER);
/*Set label font.*/
applicationtitlelabel.setFont(new Font("Verdana",Font.BOLD,14));
/*Add label to contentpane.*/
contentpane.add(applicationtitlelabel,BorderLayout.NORTH);
/*Initialize object of JButton.*/
dialbutton=new JButton("Dial");
/*Set button size.*/
dialbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
dialbutton.setActionCommand("dial");
/*Add action listener to button.*/
dialbutton.addActionListener(this);
/*Set mnemonic for button .*/
dialbutton.setMnemonic('D');
dialbutton.setToolTipText("Click this button for dialing.");
/*Initialize object of JButton.*/
cancelbutton=new JButton("Cancel");
cancelbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
cancelbutton.setEnabled(false);
cancelbutton.setActionCommand("cancel");
/*Add action listener to button.*/
cancelbutton.addActionListener(this);
/*Set mnemonic for button.*/
cancelbutton.setMnemonic('C');
/*Set tool tip text.*/
cancelbutton.setToolTipText("Click this button for cancel dialing.");
/*Initialize object of Jbutton.*/
closebutton=new JButton("Close");
/*Set button size.*/
closebutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Add action command to button.*/
closebutton.setActionCommand("close");
/*Add action listener to button.*/
closebutton.addActionListener(this);
/*Set mnemonic for button.*/
closebutton.setMnemonic('l');
/*Add tooltip text with button.*/
closebutton.setToolTipText("Click this button to close this window.");
/*Initialize object of JPanel.*/
bottompanellower=new JPanel();
/*Add buttons to panel.*/
bottompanellower.add(dialbutton);
bottompanellower.add(cancelbutton);
bottompanellower.add(closebutton);
/*Initialize object of JPanel.*/
bottompanelupper=new JPanel();
/*Initialize object of JLabel.*/
processlabel=new JLabel("Disconnected");
processlabel.setForeground(Color.blue);
/*Add label to panel.*/
bottompanelupper.add(processlabel);
/*
Initialize object of JPanel and set the layout as GridLayout.
*/
bottompanel=new JPanel(new GridLayout(2,1));
/*Add panels to bottompanel.*/
bottompanel.add(bottompanelupper);
bottompanel.add(bottompanellower);
/*Add bottompanel pane to contentpane.*/
contentpane.add(bottompanel,BorderLayout.SOUTH);
/*
Declare and initialize object of GridBagLayout.
*/
GridBagLayout gridlayout=new GridBagLayout();
```

```
/*
Declare and initialize object of GridBagConstraints.
*/
GridBagConstraints gridbagconstraints =new GridBagConstraints();
/*
Initialize object of JPanel class and set layout as GridBagLayout.
*/
middlepanel=new JPanel(gridlayout);
/*initialize object of JLabel.*/
portnamelabel=new JLabel("Port Name",JLabel.LEFT) ;
/*Set constraints for portnamelabel.*/
gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraints.anchor=GridBagConstraints.CENTER;
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridlayout.setConstraints(portnamelabel,gridbagconstraints);
/*Add label to pane.*/
middlepanel.add(portnamelabel);
/*Initialize object of JtextField.*/
portnametextbox=new JTextField();
/*Set button size.*/
portnametextbox.setPreferredSize(new Dimension(TEXTBOX_WIDTH,BUTTON_HEIGHT));
/*Set constraints for portnametextbox.*/
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
portnametextbox.setEnabled(false);
gridlayout.setConstraints(portnametextbox,gridbagconstraints);
/*Add label to pane.*/
middlepanel.add(portnametextbox);
/*Initialize object of JButton.*/
portbrowsebutton=new JButton("Browse..");
/*Add action listener to button.*/
portbrowsebutton.addActionListener(this);
/*Add action command to button.*/
portbrowsebutton.setActionCommand("portbrowse");
/*Set button size.*/
portbrowsebutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
/*Set constraints for portbrowsebutton.*/
gridbagconstraints.gridx=2;
gridbagconstraints.gridy=0;
gridlayout.setConstraints(portbrowsebutton,gridbagconstraints);
middlepanel.add(portbrowsebutton);
/*
Initialize object of JLabel and set its title.
*/
phonenumberlabel=new JLabel("Phone Number",JLabel.LEFT);
/*Set constraints for phonenumberlabel.*/
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=1;
gridbagconstraints.insets=new Insets(5,0,0,0);
gridlayout.setConstraints(phonenumberlabel,gridbagconstraints);
middlepanel.add(phonenumberlabel);
/*Initialize object of JTextField.*/
phonenumbertextbox=new JTextField();
/*Set textbox size.*/
phonenumbertextbox.setPreferredSize(new Dimension(TEXTBOX_WIDTH,BUTTON_HEIGHT));
/*
Set constraints for phonenumbertextbox.
*/
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=1;
gridlayout.setConstraints(phonenumbertextbox,gridbagconstraints);
middlepanel.add(phonenumbertextbox);
contentpane.add(middlepanel,BorderLayout.CENTER);
/*
Initialize object as PortChooser class.
*/
portchooser=new PortChooser(this);
/*
Add action listener to ok button of portchooser object.
*/
portchooser.getOkButton().addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
      portchooser.dispose();
      if (portchooser.getSelectedName()==null||portchooser.getSelectedName()=="")
      {
      System.out.println("Select the port");
      }
      else
      {
         portnametextbox.setText(portchooser.getSelectedName());
      }
   }
});
/*Set window size.*/
setSize(400,300);
```

```
/*Make it visible.*/
setVisible(true);
}
/*
actionPerformed - This method is called when the user clicks the Dial, cancel, close button.
Parameters: ae - an ActionEvent object containing details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   /*
   This is executed when user clicks the Close button.
   */
   if("close".equals(actioncommand))
   {
      System.exit(0);
   }
   /*
   This is executed when user clicks the Browse button.
   */
   if("portbrowse".equals(actioncommand))
   {
      portchooser.setVisible(true);
   }
   /*
   This is executed when user clicks the Cancel button.
   */
   if ("cancel".equals(actioncommand))
   {
      try
      {
         /*close input and output stream.*/
         is.close();
         os.close();
         /*Port close.*/
         commport.close();
         processlabel.setText("Disconnected");
      }
      catch(IOException ie)
      {
      System.err.println("Error in input output.");
      }
   }
   /*
   This is executed when user clicks the Dial button.
   */
   if("dial".equals(actioncommand))
   {
      /*
      Declare and initialize object of String class.
      */
      String portname=portnametextbox.getText();
      if (portname==null)
      {
         JOptionPane.showMessageDialog(null," Please select one port name.","
         No port choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else if (portname.trim().length()==0)
      {
         JOptionPane.showMessageDialog(null,"Please select one port name.","
         No port choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      String phoneno=phonenumbertextbox.getText();
      if (phoneno==null)
      {
         JOptionPane.showMessageDialog(null," Please enter phone number.","
         No phone number choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else if(phoneno.trim().length()==0)
      {
         JOptionPane.showMessageDialog(null," Please enter phone number.","
         No phone number choosen",JOptionPane.PLAIN_MESSAGE);
         return;
      }
      else
      {
         /*
         Check for phone number validation.
         */
         String phonenoaftertrim=phoneno.trim();
         String validstring="0123456789,";
         for(int i=0;i<phonenoaftertrim.length();i++)
         {
            if (validstring.indexOf(phonenoaftertrim.charAt(i))==-1)
```

```
                {
                   JOptionPane.showMessageDialog(null," Please enter a valid phone number.","
                   No phone number choosen",JOptionPane.PLAIN_MESSAGE);
                   return;
                }
             }
             /*Calls the portOpen()method.*/
             portOpen(phonenoaftertrim);
          }
       }
    }
public static void main(String[] args) throws IOException,
NoSuchPortException,PortInUseException,Unsuppor tedCommOperationException
{
   /*
   Declare and initialize object of PortModem class.
   */
   PortModem portmodem=new PortModem();
}
/*
portOpen - This method opens port.
Parameters: phonenoaftertrim - objcet of String class.
Return Value: NA
*/
public void portOpen(String phonenoaftertrim)
{
   /*Get selected port.*/

   commportid=portchooser.getSelectedIdentifier();
   processlabel.setText(" Dialing number...");
   switch (commportid.getPortType())
   {
      /*
      This will be executed if selected port is Serial.
      */
      case CommPortIdentifier.PORT_SERIAL:
      try
      {
         /*Try to open the port.*/
         commport=commportid.open("Modem Application",1000);
         commportid.addPortOwnershipListener(new OwnerShip());
      }
      catch(PortInUseException piu)
      {
         processlabel.setText("Port has not been opened.");
      }
      SerialPort serialport=(SerialPort)commport;
      try
      {
         /*
         Set parameters for serial port communication.
         */
         serialport.setSerialPortParams(BAUD, SerialPort.DATABITS_8,
         SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
      }
      catch(UnsupportedCommOperationException uco)
      {
         processlabel.setText("Port has not been opened.");
      }
      break;
      /*
      This will be executed if selected port is parallel.
      */
      case CommPortIdentifier.PORT_PARALLEL:
      try
      {
         /*Try to open the port.*/
         commport=commportid.open("Modem Applecation",1000);
         commportid.addPortOwnershipListener(new OwnerShip());
      }
      catch(PortInUseException piu)
      {
         processlabel.setText("Port has not been opened.");
      }
      ParallelPort parallelport=(ParallelPort)commport;
      break;
      default:
      throw new IllegalStateException("Unknown port type " + commportid);
   }
   try
   {
      /*Get Input stream.*/
      is = new DataInputStream(commport.getInputStream());
   }
   catch (IOException e)
   {
      System.err.println("Can't open input stream: write-only");
      is = null;
```

```
      }
      try
      {
         /*Get Output stream*/
         os = new PrintStream(commport.getOutputStream(), true);
      }
      catch(IOException ie)
      {
         System.err.println("Can't open output string.");
      }
      try
      {
         /*Calls send method.*/
         send("ATZ");
         /*Calls expect method.*/
         expect("OK");
         /*Calls send method.*/
         send("ATDT" + phonenoaftertrim);
      }
      catch(IOException ie)
      {
         System.err.println(ie);
      }
      try
      {
         is.close();
         os.close();
      }
      catch(IOException ie)
      {
         System.err.println(ie);
      }
   }
   /*
   class OwnerShip - This is a inner class. It implements CommPortOwnershipListener
   interface and override its methods.
   Methods:
   ownershipChange - Check to see ownership of a port.
   */
   class OwnerShip implements CommPortOwnershipListener
   {
      protected boolean owned = false;
      public void ownershipChange(int owner)
      {
         switch (owner)
         {
            /*
            Execute when end user clicks Dial button
            */
            case PORT_OWNED:
            System.out.println("An open succeeded.");
            owned = true;
            cancelbutton.setEnabled(true);
            dialbutton.setEnabled(false);
            break;
            /*
            Execute when end user clicks Cancel button
            */
            case PORT_UNOWNED:
            System.out.println("A close succeeded.");
            cancelbutton.setEnabled(false);
            dialbutton.setEnabled(true);
            owned = false;
            break;
            case PORT_OWNERSHIP_REQUESTED:
            if (owned)
            {
               if (JOptionPane.showConfirmDialog(null,"I've been asked to give up the port,
               should I?","",JOptionPane.OK_CANCEL_OPTION) == 0)
               commport.close();
            }
            else
            {
            System.out.println("Somebody else has the port");
            }
         }
      }
   }
   /*
   send - This method sends phone number to modem.
   Parameters: s - object of String class.
   Return Value: NA
   */
   protected void send(String s) throws IOException
   {
      if (start)
      {
         System.out.print(">>> ");
```

```
            System.out.print(s);
            System.out.println();
         }
         os.print(s);
         os.print("\r\n");
         if (!expect(s))
         {
            System.err.println("WARNING: Modem did not echo command.");
         }
      }
      /*
      expect - This method receives response of modem
      Parameters:   exp - objcet of String class.
      Return Value: NA
      */
      protected boolean expect(String exp) throws IOException
      {
         response = is.readLine();
         if (start)
         {
            System.out.print(response);
            System.out.println();
         }
         return response.indexOf(exp) >= 0;
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortModem class, which generates the main window of the Modem Dialer application, as shown in Figure 3-2:

Figure 3-2: The Modem Dialer Application Window

In the user interface:

- The Port Name text box: Specifies the port to which the modem is connected.
- The Phone Number text box: Specifies the phone number to which an end user wants to connect.
- The Browse button: Invokes the user interface that the PortChooser.java file creates.
- The Dial button: Enables an end user to make a phone call to the number specified in the Phone Number text box.
- The Cancel button: Ends the current call.
- The Close button: Terminates the Modem Dialer application.

The PortModem class contains an inner class called OwnerShip. The OwnerShip class implements the CommPortOwnershipListener interface provided by the JCA API and overrides the methods of the CommPortOwnershipListener interface. The various methods defined in the PortModem class are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the button that the end user clicks. When the end user clicks the Dial button, the actionPerformed() method validates the phone number specified in the Phone Number text box. In addition, the actionPerformed() method verifies the port selected in the Port Name text box. The actionPerformed() method calls the portOpen() method to open the selected port.
- `send()`: Sends the phone number specified by the end user to the modem.
- `expect()`: Receives the response of the modem through the Serial port.
- `portOpen()`: Opens the port that the end user selects from the Port Name text box. The portOpen() method creates an instance of the DataInputStream class using the input stream of the Serial port opened for communication. The DataInputStream class sends data to the modem with the help of the Serial port. The portOpen() method also creates an instance of the PrintStream class using the output stream of the Serial port opened for communication. The PrintStream class accepts the response of the modem with the help of the Serial port.

The portOpen() method also invokes the send() and expect() methods to communicate with the Serial port. The portOpen() method invokes the send() method with the ATZ parameter to check if the modem is ready for communication. If the modem responds successfully, the portOpen() method invokes the send() method with two parameters, ATDT and the phone number specified by the end user. The ATDT parameter commands the modem to dial a specified phone number.

◄ PREVIOUS NEXT ►

Selecting a Port

The PortChooser.java file creates and displays a user interface to select the port to which the modem is connected. The modem makes a phone call to the phone number specified by the end user.

Listing 3-2 shows the PortChooser.java file:

Listing 3-2: The PortChooser.java File

```
/*Imports required Comm package classes.*/
import javax.comm.*;
/*Imports required I/O package classes.*/
import java.io.*;
/*Imports required AWT classes.*/
import java.awt.*;
import java.awt.event.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
/*Imports required java.util package classes.*/
import java.util.*;
/*
class PortChooser - This class is the main class of the application.
This class initializes the interface and loads all components like the button,
combobox before displaying the result.
Constructor:
PortChooser - This constructor calls methods to create GUI and populating port list.
Methods:
   getSelectedName - Gives comm port name.
   getSelectedIdentifier - Gives comm port.
   makeGUI - Creates GUI.
populate - Generates list of serial and parallel ports
itemStateChanged: This method is called when end user select item from combobox.
*/
public class PortChooser extends JFrame implements ItemListener
{
   /*
   Declare and initialize object of HashMap class.
   */
   protected HashMap map = new HashMap();
   /*
   Declare and initialize object of CommPortIdentifier class.
   */
   protected CommPortIdentifier selectedportidentifier=null;
   /* Declare objects of JComboBox class.*/
   protected JComboBox serialportschoice;
   protected JComboBox parallelportschoice;
   /* Declare objects of JButton class.*/
   private JButton okbutton;
   private JButton cancelbutton;
   /* Declare objects of JPanel class.*/
   private JPanel bottompane;
   private JPanel centerpanel;
   /*Declare constraints.*/
   protected String selectedportname=null;
   protected final int PAD =15;
   private final int BUTTON_WIDTH=80;
   private final int BUTTON_HEIGHT=23;
   private boolean itemchangeflag=true;
   /*
   getSelectedName: It is called to get selected port name
   Parameter: NA
   Return Value: String
   */
   public String getSelectedName()
   {
      return selectedportname;
   }
   /*
   getSelectedIdentifier: It is called to get selected port.
   Parameter: NA
   Return Value: CommPortIdentifier
   */
   public CommPortIdentifier getSelectedIdentifier()
   {
      return selectedportidentifier;
   }
   public PortChooser(JFrame parent)
   {
      /*Set window title.*/
      super(" Selecting Port ");
      /*calls makeGUI method.*/
      makeGUI(parent);
      /*calls populate method.*/
      populate();
```

```
}
/*
makeGUI: This method creates GUI.
Parameter: parent - Object of JFrame.
Return Value: NA
*/
public void makeGUI(JFrame parent)
{
   /*
   Declare and initialize object of Container class.
   */
   Container cp = getContentPane();
   /*Set window's location on screen.*/
setLocation(parent.getLocation().x,parent.getLocation().y);
   /*Initialize object of JPanel.*/
   centerpanel = new JPanel();
   /*Add centerpanel to ContentPane.*/
   cp.add(BorderLayout.CENTER, centerpanel);
   /*Set layout as GridLayout.*/
   centerpanel.setLayout(new GridLayout(0, 2, PAD, PAD));
   /*
   Add object of JLabel class to ContentPane.
   */
   centerpanel.add(new JLabel("Serial Ports", JLabel.RIGHT));
   /*
   Initialize object of JComboBox class.
   */
   serialportschoice = new JComboBox();
   /*
   Add action listener to serialportschoice Combobox.
   */
   serialportschoice.addItemListener(this);
   /*
   Add serialportschoice Combobox to centerpanel panel.
   */
   centerpanel.add(serialportschoice);
   /*
   Add object of JLabel class to ContentPane.
   */
   centerpanel.add(new JLabel("Parallel Ports", JLabel.RIGHT));
   /*Initialize object of JComboBox class.*/
   parallelportschoice = new JComboBox();
   /*
   Add action listener to parallelportschoice Combobox.
   */
parallelportschoice.addItemListener(this);
   /*
   Add serialportschoice Combobox to centerpanel panel.
   */
   centerpanel.add(parallelportschoice);
   /*Initialize object of JPanel class*/
   bottompane=new JPanel();
   /*
   Initialize objects of JButton class
   */
   okbutton = new JButton("OK");
   cancelbutton = new JButton("Cancel");
   /*Set buttons size.*/
   okbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
   cancelbutton.setPreferredSize(new Dimension(BUTTON_WIDTH,BUTTON_HEIGHT));
   /*Add buttons to bottompane panel.*/
   bottompane.add(okbutton);
   bottompane.add(cancelbutton);
   /*
   Add bottompane panel to cp ContentPane.
   */
   cp.add(bottompane,BorderLayout.SOUTH);
   /* Set size of window.*/
   setSize(280,120);
   /*Add actionlistener to cancelbutton.*/
   cancelbutton.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {
      PortChooser.this.dispose();
      }
   });
}
/*
populate: This method generates ports list.
Parameter: NA
Return Value: NA
*/
public void populate()
{
   serialportschoice.addItem("");
   parallelportschoice.addItem("");
```

```
      /*
Declare object of Enumeration class and fills it with the ports obtains
by getPortIdentifiers method of CommPortIdentifier.
      */
      Enumeration pList = CommPortIdentifier.getPortIdentifiers();
      while (pList.hasMoreElements())
      {
   CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
         /*
         Put comm port in to object of hashmap class.
         */
         map.put(cpi.getName(), cpi);
         /*
         This will execute if port is serial.
         */
         if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
   serialportschoice.addItem(cpi.getName());
         }
         /*
         This will execute if port is parallel.
         */
         else if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
         {
   parallelportschoice.addItem(cpi.getName());
         }
      }
      serialportschoice.setSelectedIndex(0);
      parallelportschoice.setSelectedIndex(0);
   }
   /*
   itemStateChanged: This method is called when end user select item from combobox.
   Parameter: e - object of ItemEvent class.
   Return Value: NA
   */
   public void itemStateChanged(ItemEvent e)
   {
   if ((String)((JComboBox)e.getSource()).getSelectedItem()!="")
      {
         /*Get selected item name. */
   selectedportname = (String)((JComboBox)e.getSource()).getSelectedItem();
   selectedportidentifier = (CommPortIdentifier)map.get(selectedportname);
   if (serialportschoice.equals((JComboBox)e.getSource()))
         {
   if (parallelportschoice.getSelectedIndex()>0)
            {
   parallelportschoice.setSelectedIndex(0);
            }
         }
         else
         {
   serialportschoice.setSelectedIndex(0);
         }
      }
   }
   /*
   getOkButton: This method returns an object of OK button.
   Parameter: NA
   Return Value: JButton
   */
   public JButton getOkButton()
   {
      return okbutton;
   }
}
```

Download this listing.

The above listing displays a user interface that lists the ports on a computer. From this list, the end user can select the port to which the modem is connected. The PortChooser.java file uses the CommPortIdentifier class of javax.comm to retrieve the list of ports on a computer.

The user interface of the PortChooser.java file appears when an end user clicks Browse in the user interface of the Modem Dialer application. Figure 3-3 shows the user interface of the PortChooser.java file:

Figure 3-3: The User Interface of the PortChooser.java File

In the PortChooser.java file, the default constructor of the PortChooser class accepts an object of the PortModem class as an input parameter. The various methods defined in the PortChooser class are:

- `makeGUI()`: Creates the user interface to select the Serial port to which the modem is connected.
- `Populate()`: Generates a list of serial and parallel ports.
- `itemStateChanged()`: Retrieves the item that is currently selected in the Serial Ports or Parallel Ports drop-down list box.
- `getSelectedName()`: Returns the name of the selected Serial port.
- `getSelectedIdentifier()`: Returns an object of the CommPortIdentifier class that represents the selected Serial port.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the button that the end user clicks. When the end user clicks the OK button, the actionPerformed() method opens the selected port . If the end user clicks the Cancel button, the actionPerformed() method closes the user interface of the PortChooser.java file.

PREVIOUS NEXT

Unit Testing

To test the Modem Dialer application:

1. Download the Javacomm20-win32 JCA, which is available in zip format. The JCA is available at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the PortModem.java and PortChooser.java files to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Run the Modem Dialer application using the following command at the command prompt:

   ```
   java PortModem
   ```

10. Click the Browse button in the Modem Dialer window. The user interface of the PortChooser.java file opens, as shown in Figure 3-3.
11. Select the port to which the modem is connected and click the OK button. This opens the port selected by the end user to make the phone call.
12. Specify the phone number to which the call is to be made in the Phone Number text box. Figure 3-4 shows the user interface of the Modem Dialer application with the specified port and phone number:

Figure 3-4: Making a Phone Call

13. Click the Dial button. This makes a call to the number specified in the Phone Number text box.
14. Click the Cancel button to end the call.
15. Click the Close button to terminate the Modem Dialer application.

Chapter 4: Creating a Printing Application

The Java Communication API (JCA) supports the javax.comm package. This package enables you to use the CommPortIdentifier and CommPort classes to detect the printer port. The CommPortIdentifier class provides a list of the ports available on your computer and the CommPort class opens a selected port for input/output operations.

This chapter explains how to develop a Printing application that uses the javax.comm package to select the printer port and print the text in a document. In addition, the application allows an end user to select and change the font and the color of the text in the document.

Architecture of the Printing Application

An end user can use the Printing application to list the ports on a computer. The end user can select the printer port, and open and print any text document. The end user can also change the font and the color of selected text in the document.

The Port Communication application uses the following files:

- CommPrinting.java: Creates the user interface of the Printing application. This is the main file of the application.
- ColorChoose.java: Enables the end user to use the Printing application to select the color of the text in the document to be printed.
- FontChoose.java: Enables the end user to use the Printing application to select the font of the text in the document to be printed.
- PortChoice.java: Lists the ports on a computer and enables the end user to select the port to which the printer is connected.
- PrintComponent_Class.java: Specifies print setup properties and prints the text in the document that is currently open in the Printing application.

Figure 4-1 shows the architecture of the Printing application:

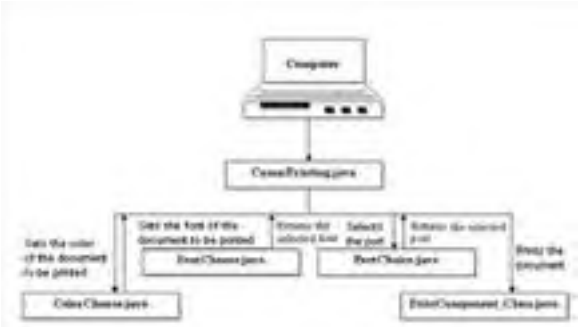

Figure 4-1: Architecture of the Printing Application

In the Printing application, the CommPrinting.java file displays a user interface that consists of a menu bar and an edit pane. The various menu options on the menu bar enable an end user to specify the color and the font of the text in the document to be printed.

The CommPrinting.java file invokes:

- The ColorChoose.java file when the end user selects the Color menu item from the Color menu.
- The FontChoose.java file when the end user selects the Font menu option from the Font menu of the Printing application.
- The PortChoice.java file when the end user selects the Choose COM Port to Print menu option from the File menu.
- The PrintComponent_Class file to print the text in the document using the Printing application.

Creating the User Interface

The CommPrinting.java file is the main file of the Printing application. This file displays a user interface with a menu bar and an edit pane. The menu bar contains three menus: File, Font, and Color. The menu options of these menus enable end users to open a document, select the color and the font of the text in the document, select the printer port, specify page setup, and print the text in the document.

Listing 4-1 shows the CommPrinting.java file:

Listing 4-1: The CommPrinting.java File

```
/* Imports required Comm classes */
import javax.comm.*;
/* Imports required I/O classes */
import java.io.*;
/* Imports required AWT classes */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/* Imports required Filedialogbox classes */
import javax.swing.filechooser.*;
/* Imports required Print classes */
import javax.print.*;
import javax.print.attribute.*;
import javax.print.event.*;
import java.awt.print.*;
import java.lang.reflect.*;
import java.awt.Graphics2D;
import javax.print.attribute.standard.*;
/*
class CommPrinting - This class is the main class of the application. This class initializes
the interface and loads all components like the menubar,editpane before displaying the result.
   Constructor:
CommPrinting-This constructor creates GUI.
   Methods:
         createGUI-This method creates GUI.
addlEvent - This method add event to components
actionPerformed - This method describes which action will be performed on which component.
getExtension - This method gives file extension.
printDoc - This method print document.
main - This method creates the main window of the application and displays all the components.
*/
class CommPrinting extends JFrame  implements ActionListener
{
   /*
      Declare object of JMenuBar class.
   */
   JMenuBar menubar;
   /*
      Declare objects of JMenu class.
   */
    JMenu file_menu;
   JMenu setting_menu;
   /*
      Declare objects of JMenuItem class.
   */
JMenuItem print_menuitem, setup_menuitem,  exit_menuitem, open_menuitem,font_menuitem,color_menuitem
   /*
      Declare object of JFileChooser class.
   */
   JFileChooser filechooser;
   /*
      Declare object of JScrollPane class.
   */
   JScrollPane scrollpane;
   /*
      Declare object of JTextPane class.
   */
   JTextPane textpane;
   /*
      Declare object of DataInputStream class.
   */
   protected DataInputStream is;
   /*
      Declare object of PrintStream class.
   */
   protected PrintStream os;
   /*
      Declare object of InputStream class.
   */
   protected InputStream istr;
   /*
      Declare object of CommPort class.
   */
```

```
CommPort commport;
/*
Declare and Initialize object of FontChoose class.
*/
public FontChoose font = new FontChoose();
/*
Declare and Initialize object of ColorChoose class.
*/
public ColorChoose color=new  ColorChoose();
/*
Declare and Initialize object of PortChoice class.
*/
PortChoice comport=new PortChoice(null,this);
private boolean PrintJobDone = false;
public static final int TIMEOUTSECONDS = 30;
public static final int BAUD = 9600;
/*
Main method that creates the instance of the CommPrinting class.
*/
public static void main(String[] args) throws IOException, NoSuchPortException, _
PortInUseException, UnsupportedCommOperationException
{
   CommPrinting commprint=new CommPrinting();
}
public CommPrinting()
{
   super("Printing Application");
   createGUI();
   addlEvent();
}
/*
   createGUI: It is called to create menubar and textpane.
   Parameter: N/A
   Return Value: N/A
*/
public void createGUI()
{
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   Container container=getContentPane();
   /*
   Set the layout of the frame as BorderLayout.
   */
   container.setLayout(new BorderLayout());
   /*
   Initialize the object of JMenuBar class.
   */
   menubar = new JMenuBar();
   /*
   Initialize the objects of JMenu class and set the Title.
   */
        file_menu = new JMenu("File");
   setting_menu=new JMenu("Settings");
   /*
   Initialize the objects of JMenuItem class and set the Title.
   */
        open_menuitem = new JMenuItem("Open");
         setup_menuitem = new JMenuItem("Page Setup & Print");
         print_menuitem = new JMenuItem("Choose COM Port to Print");
        exit_menuitem = new JMenuItem("Exit");
        font_menuitem = new JMenuItem("Font");
   color_menuitem=new JMenuItem("Color");
   /*
   Initialize the object of JTextPane class.
   */
   textpane=new JTextPane();
   /*
   Initialize the object of JScrollPane class and set this on textpane object.
   */
   scrollpane=new JScrollPane(textpane);
   /*
   Add scrollpane to content pane and place it in center of container.
   */
   container.add(scrollpane, BorderLayout.CENTER);
   /*
   Add menuitems to menu and set actioncommand to them.
   */
   file_menu.add(open_menuitem);
```

```
   file_menu.add(setup_menuitem);
   file_menu.add(print_menuitem);
   file_menu.add(exit_menuitem);
   setting_menu.add(font_menuitem);
   setting_menu.add(color_menuitem);
   open_menuitem.setActionCommand("Open");
   setup_menuitem.setActionCommand("Page");
   setup_menuitem.setEnabled(false);
   print_menuitem.setActionCommand("Print");
   exit_menuitem.setActionCommand("Exit");
   font_menuitem.setActionCommand("font");
   color_menuitem.setActionCommand("color");
   /*
      Add menu to menubar.
   */
   menubar.add(file_menu);
   menubar.add(setting_menu);
   setJMenuBar(menubar);
   /*
   Initialize object of JFileChooser class.
   */
   filechooser=new JFileChooser();
   /*
      Set filter with file dialog box.
   */
   filechooser.setFileFilter(new javax.swing.filechooser.FileFilter ()
  {
      public boolean accept(File f)
      {
          if (f.isDirectory())
          {
           return true;
          }
          String name = f.getName();
          if (name.endsWith(".txt"))
          {
           return true;
          }
          return false;
   }
   public String getDescription()
   {
      return ".txt";
         }
     });
   /*
   Set the location at which window will be displayed.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   /*
      Set window's size.
   */
   setSize(500,500);
   setVisible(true);
}
/*
addlEvent: It is called to add event listener with components.
      Parameter: N/A
      Return Value: N/A
*/
public void addlEvent()
{
   addWindowListener(new WindowAdapter(){public void windowClosing(WindowEvent e){System.exit(0);
   open_menuitem.addActionListener(this);
   setup_menuitem.addActionListener(this);
   print_menuitem.addActionListener(this);
   exit_menuitem.addActionListener(this);
   font_menuitem.addActionListener(this);
   color_menuitem.addActionListener(this);
}
/*
actionPerformed: It is called when user clicks open, Print setup &
print, print,Exit,font or color menu item.
Parameter: ae - an ActionEvent object containing details of the event.
Return Value: N/A
*/
 public void actionPerformed(ActionEvent ae)
 {
   /*
   This is executed when user clicks the open menuitem.
   */
   if ("Open".equals(ae.getActionCommand()))
   {
      int returnVal = filechooser.showOpenDialog(this);
      if(returnVal == JFileChooser.APPROVE_OPTION)
      {
         String fileextension=getExtension(filechooser.getSelectedFile());
```

```
            String fileabspath=filechooser.getSelectedFile().getAbsolutePath();
            if (fileextension.equals("jpg")||fileextension.equals("JPG")||
            fileextension.equals("gif")||fileextension.equals("GIF")||
            fileextension.equals("png")||fileextension.equals("PNG"))
            {
               textpane.insertIcon(new ImageIcon(fileabspath));
            }
            else
            {
               try
               {
                  FileInputStream inputStream = new
                  FileInputStream(filechooser.getSelectedFile().getAbsolutePath());
                  BufferedReader br = new BufferedReader(new InputStreamReader(inputStream));
                  StringBuffer strOut=new StringBuffer();
                  String strTemp="";
                  while ((strTemp = br.readLine())!=null)
                  {
                     strOut.append(strTemp).append("\n");
                  }
                  textpane.setText(strOut.toString());
               }
               catch(IOException e)
               {
                  System.out.println("Error in file reading"+e);
               }
               }
            }
         }
         /*
         This is executed when user clicks the Print menuitem.
         */
         if ("Print".equals(ae.getActionCommand()))
         {
            /*
            Declare object of PortChoice class.
            */
            comport.setVisible(true);
            setup_menuitem.setEnabled(true);
         }
         /*
         This is executed when user clicks the page setup and print menuitem.
         */
         if ("Page".equals(ae.getActionCommand()))
         {
            new PrintComponent_Class(textpane).pageSetupAndPrint();
         }
         /*
         This is executed when user clicks the exit menuitem.
         */
         if ("Exit".equals(ae.getActionCommand()))
         {
            System.exit(0);
         }
         /*
         This is executed when user clicks the font menuitem.
         */
         if ("font".equals(ae.getActionCommand()))
         {
            font.setVisible(true);
            font.getOk().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  textpane.setFont(font.font());
                  font.setVisible(false);
               }
            });
            font.getCancel().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  font.setVisible(false);
               }
            });
         }
         /*
         This is executed when user clicks the color menuitem.
         */
         if ("color".equals(ae.getActionCommand()))
         {
            color.pack();
            color.setVisible(true);
            color.getOk().addActionListener(new ActionListener()
            {
               public void actionPerformed(ActionEvent ae)
               {
                  textpane.setForeground(color.color());
```

```
            color.setVisible(false);
         }
      });

      color.getCancel().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            color.setVisible(false);
         }
      });
   }
}
/*
getExtension:This method will extract file extension from file.
Parameter:f - object of File class
Return Value: file extension
*/
public String getExtension(File f)
{
   if(f != null)
   {
      String filename = f.getName();
      int i = filename.lastIndexOf('.');
      if(i>0 && i<filename.length()-1)
      {
         return filename.substring(i+1).toLowerCase();
      };
   }
   return null;
}
/*
printDoc:This method will extract file extension from file.
Parameter:commportname - name of selected com port.
commid - object of CommPortIdentifier.
Return Value: NA
*/
public void printDoc(String commportname,CommPortIdentifier commid)
{
   if (commportname == null)
   {
      JOptionPane.showMessageDialog(null,"No port selected.","Port Choose",JOptionPane.PLAIN_M
   }
   else
   {
      try
      {
         commport=commid.open("DarwinSys DataComm", TIMEOUTSECONDS * 100);
      }
      catch(Exception e)
      {
      }
      ParallelPort pPort = (ParallelPort)commport;
      int mode = pPort.getMode();
      switch (mode)
      {
            case ParallelPort.LPT_MODE_ECP:
            System.out.println("Mode is: ECP");
            break;
            case ParallelPort.LPT_MODE_EPP:
            System.out.println("Mode is: EPP");
            break;
            case ParallelPort.LPT_MODE_NIBBLE:
            System.out.println("Mode is: Nibble Mode");
            break;
            case ParallelPort.LPT_MODE_PS2:
            System.out.println("Mode is: Byte mode.");
            break;
            case ParallelPort.LPT_MODE_SPP:
            System.out.println("Mode is: Compatibility mode.");
            break;
            default:
            throw new IllegalStateException
            ("Parallel mode " + mode + " invalid.");
         }
         try
         {
            is = new DataInputStream(commport.getInputStream());
            setup_menuitem.setEnabled(true);
         }
         catch (IOException e)
         {
            System.err.println();
            is = null;
         }
         try
         {
            os = new PrintStream(commport.getOutputStream(), true);
```

```
                }
                catch(IOException e)
                {
                   JOptionPane.showMessageDialog(null,"Can't open output stream.","
                   Port Choose",JOptionPane.PLAIN_MESSAGE);
                }
                int startpos=0;
                int endpos=0;
                StringBuffer srtbuffer=new StringBuffer(2500);
                srtbuffer=new StringBuffer(textpane.getText());
                endpos=srtbuffer.indexOf("\n",startpos);
                while (endpos!=-1)
                {
                   String strprint =  srtbuffer.substring(startpos,endpos) ;
                   os.println(strprint);
                   startpos=endpos+1;
                   endpos=srtbuffer.indexOf("\n",startpos);
                }
                os.close();
                commport.close();
                comport.setVisible(false);
            }
        }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the CommPrinting class. This class generates the main window of the Printing application, as shown in Figure 4-2:

Figure 4-2: The Printing Application User Interface

The edit pane enables an end user to display and edit the document to be printed.

The menu bar provides two menus, File and Settings. The File menu of the Printing application provides four menu options: Open, Page Setup & Print, Choose COM Port to Print, and Exit, as shown in Figure 4-3:

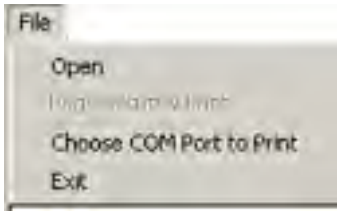

Figure 4-3: The File Menu

The Open menu option of the File menu enables the end user to open the text document to be printed. The user interface of the Printing application provides a file filter to ensure that the end user selects and opens only the text file. The Page Setup and Print menu option on the File menu specifies page setup properties and prints the text in the document. The Choose COM Port to Print menu option on the File menu allows the end user to select the COM port to which the printer is connected. The Exit menu option terminates the Printing application.

The Settings menu of the Printing application provides two menu options, Font and Color, as shown in Figure 4-4:

Figure 4-4: The Color Menu

The Font menu option enables an end user to select the font of the text in the document to be printed. The Color menu option specifies the color of the text in the document printed using the Printing application.

PREVIOUS NEXT

Selecting a Port

The PortChoice.java file allows an end user to display and select the port used to print the text in the document. The PortChoice.java file creates a user interface that displays the list of ports on a computer. The end user can select the port to which the printer is connected from the list of ports. The user interface of the PortChoice.java file appears when the end user selects the Choose COM Port to Print menu option on the File menu of the Printing application.

Listing 4-2 shows the PortChoice.java file:

Listing 4-2: The PortChoice.java File

```
/* Imports required Comm classes */
import javax.comm.*;
/* Imports required I/O classes */
import java.io.*;
/* Imports required AWT classes  */
import java.awt.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/* Imports required Util classes */
import java.util.*;
/*
class PortChoice - This class is the main class of the application. This class initializes
the interface and loads all components like the dropdown listbox, button
before displaying the result.
   Constructor:
      PortChoice-This constructor creates GUI.
   Methods:
getPortDescription - This method gives port description.
openPort - This method opens a selected port.
getSerialPorts - This method fill list of serial ports.
getCommPorts - This method fill list of comm ports.
emptyList - This method clear a list.
main - This method creates the main window of the application and displays all the components.
*/
public class PortChoice extends JDialog implements ItemListener
{
   protected String selectedPortName;
   /*
      Declare object of JButton class.
   */
   JButton okButton;
   JButton cancelButton;
   /*
      Declare object of JComboBox class.
   */
   protected JComboBox commChoice;
   /*
      Declare objects of JLabel class.
   */
   JLabel frametextlabel;
   JLabel parallelportlabel;
   /*
   Declare object of CommPortIdentifier class.
   */
   protected CommPortIdentifier selectedPortIdentifier;
   /*
      Declare object of CommPrinting class.
   */
   CommPrinting commprint;
   protected final int PAD = 5;
   /*
   itemStateChanged: This method is called where end user select item from dropdown listbox.
   Parameter: e - an ItemEvent object containing details of the event.
   Return Value: N/A
   */
   public void itemStateChanged(ItemEvent e)
   {
      selectedPortName = (String)((JComboBox)e.getSource()).getSelectedItem();
      try
      {
         selectedPortIdentifier = javax.comm.CommPortIdentifier.getPortIdentifier(selectedPortName);
      }
      catch(NoSuchPortException pe)
      {
         System.out.println("No port exists"+pe);
      }
   }
   /*
   itemStateChanged: This method is called to get selected port name.
   Parameter: N/A
```

```
Return Value: String
*/
public String getSelectedName()
{
   return selectedPortName;
}
/*
itemStateChanged: This method is called to get selected port.
Parameter: N/A
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getSelectedIdentifier()
{
   return selectedPortIdentifier;
}
public PortChoice(JFrame parent, CommPrinting cprint)
{
   super(parent, "COM Port Selection", true);
   commprint=cprint;
   makeGUI();
   populate();
   finishGUI();
}
/*
makeGUI: This method is called to create GUI.
Parameter: N/A
Return Value: N/A
*/
protected void makeGUI()
{
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Exception in setting WLAF"+e);
   }
   Container cp = getContentPane();
   setResizable(false);
   /*
   Initialize the object of JLabel class.
   */
   frametextlabel=new JLabel("Choose COM Port to Print");
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   /*
   Set the location at which window will be displayed.
   */
   setLocation((scrnSize.width / 2) - 350, (scrnSize.height / 2) - 250);
   frametextlabel.setHorizontalAlignment(JLabel.CENTER);
   frametextlabel.setFont(new Font("Verdana", Font.BOLD, 14));
   cp.add(BorderLayout.NORTH,frametextlabel);
   /*
   Declare object of centerPanel class and initialize it.
   */
   JPanel centerPanel = new JPanel();
   cp.add(BorderLayout.CENTER, centerPanel);
   /*
      Set frame size.
   */
   setSize(250,110);
   GridBagLayout gblayout=new GridBagLayout();
   /*
      Set frame layout to GridBagLayout.
   */
   centerPanel.setLayout(gblayout);
   GridBagConstraints gbconstraints=new GridBagConstraints();
   gbconstraints.fill=GridBagConstraints.HORIZONTAL;
   gbconstraints.gridx=0;
   gbconstraints.gridy=0;
   gbconstraints.weightx=1.0;
   gbconstraints.weighty=1.0;
   gbconstraints.anchor=GridBagConstraints.CENTER;
   parallelportlabel=new JLabel("COM Ports");
   parallelportlabel.setHorizontalAlignment(JLabel.CENTER);
   parallelportlabel.setFont(new Font("Verdana", Font.PLAIN, 12));
   gblayout.setConstraints(parallelportlabel,gbconstraints);
   centerPanel.add(parallelportlabel);
   gbconstraints.gridx=1;
   gbconstraints.gridy=0;
   gbconstraints.weightx=1.0;
   gbconstraints.weighty=1.0;
   gbconstraints.gridwidth=1;
   gbconstraints.gridheight=1;
   gbconstraints.ipadx=0;
```

```
        gbconstraints.ipady=0;
        gbconstraints.anchor=GridBagConstraints.WEST;
        commChoice = new JComboBox();
        gblayout.setConstraints(commChoice,gbconstraints);
        centerPanel.add(commChoice);
        commChoice.setEnabled(false);
        addWindowListener(new WindowAdapter()
        {
           public void windowClosed(WindowEvent we)
           {
              PortChoice.this.dispose();
           }
        });
        JPanel jbuttonpanel=new JPanel(new FlowLayout());
        jbuttonpanel.add(okButton = new JButton("OK"));
        jbuttonpanel.add(cancelButton = new JButton("Cancel"));
        cp.add(BorderLayout.SOUTH, jbuttonpanel);
        okButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent e)
           {
              commprint.printDoc(getSelectedName(),getSelectedIdentifier());
           }
        });
        cancelButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              setVisible(false);
           }
        });
     }
     /*
     populate: This method is called to populate COM port in dropdown listbox.
     Parameter: N/A
     Return Value: N/A
     */
     protected void populate()
     {
        Enumeration pList = CommPortIdentifier.getPortIdentifiers();
        while (pList.hasMoreElements())
        {
           CommPortIdentifier cpi = (CommPortIdentifier)pList.nextElement();
           if (cpi.getPortType() == CommPortIdentifier.PORT_PARALLEL)
           {
              commChoice.setEnabled(true);
              commChoice.addItem(cpi.getName());
           }
        }
        commChoice.setSelectedIndex(-1);
     }
     /*
     finishGUI: This method is called to add event listener to dropdown listbox
     Parameter: N/A
     Return Value: N/A
     */
     protected void finishGUI()
     {
        commChoice.addItemListener(this);
     }
}
```

Download this listing.

The above listing allows an end user to select the port used to print the text in the document. The PortChoice.java file uses the CommPortIdentifier class of the javax.comm package to retrieve the list of ports on a computer. Figure 4-5 shows the interface of the PortChoice.java file:

Figure 4-5: The Interface of the PortChoice.java File

Selecting Color

The ColorChoose.java file allows end users to select the color of the text in the document to be printed. The user interface of the ColorChoose.java file appears when an end user clicks the Color menu item from the Color menu of the Printing application.

Listing 4-3 shows the ColorChoose.java file:

Listing 4-3: The ColorChoose.java File

```
/* Import javax.swing package classes */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JSlider;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
import java.awt.Dimension;
/* Import java.awt package classes */
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Color;
/* Import javax.swing.event package classes */
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
/*
class ColorChoose - This class creates a Color dialog box that enables the
end user to change the color of the file text.
      Fields:
      panel - Contains all the components of the Color dialog
      redLabel - Contains the content of Red label
      greenLabel - Contains the content of Green label
      blueLabel - Contains the content of Blue label
      previewLabel - Contains the content of Preview label
      redSlider - Enables the end user to select the red value
      greenSlider - Enables the end user to select the green value
      blueSlider - Enables the end user to select the blue value
      labelText - Contains the content of preview text
         ok - Creates an OK button
         cancel - Creates a cancel button
         r = 0 - Stores the Red value
         g = 0 -   Stores the Green value
         b = 0 - Stores the Blue value
      Methods:
      getOK() - This method returns the OK button object
      getCancel() - This method returns the Cancel button object
      stateChanged() - This method is invoked when an end user slide slider
      color() - This method returns the color
*/
public class ColorChoose extends JDialog implements ChangeListener
{
   /* Declare the object of JPanel class */
   JPanel panel;
   /* Declare the objects of JLabel class */
   JLabel redLabel;
   JLabel greenLabel;
   JLabel blueLabel;
   JLabel previewLabel;
   /* Declare the objects of JSlider class */
   JSlider redSlider;
   JSlider greenSlider;
   JSlider blueSlider;
   /* Declare the object of JTextField class */
   JTextField labelText;
   /* Declare the objects of JButton class */
   JButton ok;
   JButton cancel;
   /* Declare the integer for storing the RGB values */
   int r = 0;
   int g = 0;
   int b = 0;
   /* Default constructor */
   public ColorChoose()
   {
      /* Set the title of the Font dialog */
      setTitle("Selecting the Color");
      /* Set resizable button to FALSE */
      setResizable(false);
      /* Initialize the object of JPanel */
      panel = new JPanel();
      /* Set the Layout as GridLayout*/
      panel.setLayout(new GridLayout(5,2,1,1));
      /* Add the panel to Color dialog frame */
```

```
        getContentPane().add(panel);
        /* Initialize and add Red label to the panel */
        redLabel = new JLabel("Red: ");
        panel.add(redLabel);
        /* Initialize and add Red slider to the panel */
        redSlider = new JSlider(0, 255, 1);
        panel.add(redSlider);
        /* Set Border to the Red slider */
        redSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Red slider */
        redSlider.addChangeListener(this);
        /* Initialize and add Green label to the panel */
        greenLabel = new JLabel("Green: ");
        panel.add(greenLabel);
        /* Initialize and add Green slider to the panel */
        greenSlider = new JSlider(0, 255, 1);
        panel.add(greenSlider);
        /* Set Border to the Green slider */
        greenSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Green slider */
        greenSlider.addChangeListener(this);
        /* Initialize and add Blue label to the panel */
        blueLabel = new JLabel("Blue: ");
        panel.add(blueLabel);
        /* Initialize and add Blue slider to the panel */
        blueSlider = new JSlider(0, 255, 1);
        panel.add(blueSlider);
        /* Set Border to the Blue slider */
        blueSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Add state change listener to Blue slider */
        blueSlider.addChangeListener(this);
        /* Initialize and add Preview label to the panel */
        previewLabel = new JLabel("Preview: ");
        panel.add(previewLabel);
        /* Initialize and add Preview text field to the panel */
        labelText = new JTextField(10);
        panel.add(labelText);
        /* Initialize and add OK button to the panel */
        JPanel pan=new JPanel(new FlowLayout(FlowLayout.CENTER));
        ok = new JButton("OK");
        ok.setPreferredSize(new Dimension(80,23));
        pan.add(ok);
        /* Initialize and add Cancel button to the panel */
        cancel = new JButton("Cancel");
        cancel.setPreferredSize(new Dimension(80,23));
        pan.add(cancel);
        panel.add(new JLabel(" "));
        panel.add(pan);
    }
    /*
getOk() method - This method is invoked when end user click the OK button of the Color dialog box
          parameter - NA
          return value - ok
    */
    public JButton getOk()
    {
        return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel button of the Color di
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
        return cancel;
    }
    /*
    stateChanged() - This method is called when the user slides any slider of the Color dialog box.
    Parameters:   ce - an ChangeEvent object containing details of the event.
    Return Value: NA
     */
    public void stateChanged(ChangeEvent ce)
    {
        /* This section is executed, when end user slides the Red slider */
        if(ce.getSource() == redSlider)
        {
            r = redSlider.getValue();
        }
        /* This section is executed, when end user slides the Green slider */
        else if(ce.getSource() == greenSlider)
        {
            g = greenSlider.getValue();
        }
        /* This section is executed, when end user slides the Blue slider */
        else if(ce.getSource() == blueSlider)
        {
            b = blueSlider.getValue();
```

```
        }
        /* Create the object of Color class */
        Color c = new Color(r, g, b);
        /* Set the background color of the preview text field */
        labelText.setBackground(c);
    }
    /*
    color() - This method is set color of the file text
    Parameters:   NA
    Return Value: color
     */
    public Color color()
    {
        Color color = new Color(redSlider.getValue(), greenSlider.getValue(), blueSlider.getValue());
        return color;
    }
}
```

Download this listing.

The ColorChoose.java file provides a user interface with three sliders and two buttons. In addition, the user interface of the ColorChoose.java file provides a text box that shows a preview of the selected color. The end user can use the three sliders on the user interface of ColorChoose.java to specify the RGB value of the selected color. The OK button on the user interface sets the color of the text in the document to be printed to the selected color. The Cancel button on the user interface cancels the current selection. Figure 4-6 shows the user interface of the ColorChoose.java file:

Figure 4-6: The Selecting the Color Dialog Box

Clicking any button on the user interface of ColorChoose.java calls the actionPerformed() method. This method acts as a listener for the events that the end user generates.

Selecting Font

The FontChoose.java file allows an end user to select the font of the text in the document to be printed. The user interface of the FontChoose.java file appears when the end user clicks the Font menu item on the Font menu of the Printing application.

Listing 4-4 shows the FontChoose.java file:

Listing 4-4: The FontChoose.java File

```
/* Import javax.swing package classes */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Import java.awt package classes */
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/* Import javax.swing.event package classes */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.*;
/*
class FontChoose - This class creates a Font dialog box that enables the end
user to change the font family, size and type of the text.
      Fields:
      fontLabel - Contains the content of Font label
      sizeLabel - Contains the content of Size label
      typeLabel - Contains the content of Type label
      previewLabel - Contains the content of preview
      label - Contains the preview contents
      fontText - Contains the selected font family name
      typeText - Contains the selected font type name
      sizeText - Contains the selected font size name
      fontScroll - Contains the Font list
      typeScroll - Contains the Type list
      sizeScroll - Contains the Size list
      fontList - Contains all the available font family
      typeList - Contains all the available types of font style
      sizeList - Contains all the available font size
      ok - Creates an OK button
      cancel - Creates a cancel button
      Methods:
      getOK() - This method returns the OK button object
      getCancel() - This method returns the Cancel button object
      valueChanged() - This method is invoked when an end user select the item from the List box.
      font() - This method returns the font
*/
public class FontChoose extends JDialog implements ListSelectionListener
{
   /* Declare the objects of JPanel class */
   JPanel pan1;
   JPanel pan2;
   JPanel pan3;
   /* Declare the objects of JLabel class */
   JLabel fontLabel;
   JLabel sizeLabel;
   JLabel typeLabel;
   JLabel previewLabel;
   /* Declare the objects of JTextField class */
   JTextField label;
   JTextField fontText;
   JTextField sizeText;
   JTextField typeText;
   /* Declare the objects of JScrollPane class */
   JScrollPane fontScroll;
   JScrollPane typeScroll;
   JScrollPane sizeScroll;
   /* Declare the objects of JList class */
   JList fontList;
   JList sizeList;
   JList typeList;
   /* Declare the objects of JButton class */
   JButton ok;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /*
```

```
getOk() method - This method is invoked when end user click the OK button of the Font dialog box
parameter - NA
return value - ok
*/
public JButton getOk()
{
   return ok;
}
/*
getCancel() method - This method is invoked when end user click the Cancel button of the Font dia
parameter - NA
return value - cancel
*/
public JButton getCancel()
{
   return cancel;
}
/* Define the default constructor */
public FontChoose()
{
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   /* Set the title of the Font dialog */
   setTitle("Selecting the Font");
   /* Set the size of Font dialog */
   setSize(300, 400);
   /* Set resizable button to FALSE */
   setResizable(false);
   /* Initialize the object of GridBagLayout */
   gbl = new GridBagLayout();
   /* Set the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of GridBagConstraints class */
   gbc = new GridBagConstraints();
   /* Initialize the Font label object and add it to the 1,1,1,1 position with WEST alignment */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontLabel = new JLabel("Fonts: ");
   getContentPane().add(fontLabel, gbc);
   /* Initialize the Size label object and add it to the 2,1,1,1 position with WEST alignment */
   gbc.gridx = 2;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeLabel = new JLabel("Sizes: ");
   getContentPane().add(sizeLabel, gbc);
   /* Initialize the Types label object and add it to the 3,1,1,1 position with WEST alignment */
   gbc.gridx = 3;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeLabel = new JLabel("Types: ");
   getContentPane().add(typeLabel, gbc);
   /* Initialize the Font text field object and add it to the 1,2,1,1 position with WEST alignmen
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontText = new JTextField("Arial", 12);
   getContentPane().add(fontText, gbc);
   /* Initialize the Size text field object and add it to the 2,2,1,1 position with WEST alignmen
   gbc.gridx = 2;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeText = new JTextField("8", 4);
   getContentPane().add(sizeText, gbc);
   /* Initialize the Types text field object and add it to the 3,2,1,1 position with WEST alignme
   gbc.gridx = 3;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeText = new JTextField("Regular", 6);
   getContentPane().add(typeText, gbc);
   /* Initialize the Font list object and add it to the Font scroll pane object.
```

```
    Add this scroll pane object to 1,3,1,1 position with WEST alignment */
    gbc.gridx = 1;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames
    fontList = new JList(fonts);
    fontList.setFixedCellWidth(110);
    fontList.addListSelectionListener(this);
    fontList.setSelectedIndex(0);
    fontScroll = new JScrollPane(fontList);
    getContentPane().add(fontScroll, gbc);
    /* Initialize the Size list object and add it to the Size scroll pane object.
    Add this scroll pane object to 2,3,1,1 position with WEST alignment */
    gbc.gridx = 2;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] sizes = {"8", "10", "12", "14", "16", "18", "20", "24", "28", "32", "48", "72"};
    sizeList = new JList(sizes);
    sizeList.setSelectedIndex(0);
    sizeList.setFixedCellWidth(20);
    sizeList.addListSelectionListener(this);
    sizeScroll = new JScrollPane(sizeList);
    getContentPane().add(sizeScroll, gbc);
    /* Initialize the types list object and add it to the Types scroll pane object.
    Add this scroll pane object to 3,3,1,1 position with WEST alignment. */
    gbc.gridx = 3;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.WEST;
    String[] types = {"Regular", "Bold", "Italic", "Bold Italic"};
    typeList = new JList(types);
    typeList.setFixedCellWidth(60);
    typeList.addListSelectionListener(this);
    typeList.setSelectedIndex(0);
    typeScroll = new JScrollPane(typeList);
    getContentPane().add(typeScroll, gbc);
    /* Initialize the preview label and add it to 1,4,3,1 position with CENTER alignment */
    gbc.gridx = 1;
    gbc.gridy = 4;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan1 = new JPanel();
    pan1.setLayout(new FlowLayout());
    previewLabel = new JLabel("Preview:");
    pan1.add(previewLabel);
    getContentPane().add(pan1, gbc);
    /* Initialize the preview text field and add it to 1,5,3,1 position with CENTER alignment */
    gbc.gridx = 1;
    gbc.gridy = 5;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan2 = new JPanel();
    pan2.setLayout(new FlowLayout());
    label = new JTextField("AaBaCcDdeEfFgGhHjJ");
    label.setEditable(false);
    label.setBorder(BorderFactory.createEtchedBorder());
    label.setFont(new Font("Verdana",Font.PLAIN,20));
    pan2.add(label);
    getContentPane().add(pan2, gbc);
    /* Initialize the OK and Cancel button. Add these two buttons to the panel. Set layout of
    the panel to FlowLayout. Now add this panel to the 1,6,4,1 position with CENTER alignment. */
    gbc.gridx = 1;
    gbc.gridy = 6;
    gbc.gridwidth = 3;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan3 = new JPanel();
    pan3.setLayout(new FlowLayout());
    ok = new JButton("OK");
    cancel = new JButton("Cancel");
    pan3.add(ok);
    pan3.add(cancel);
    getContentPane().add(pan3, gbc);
}
/*
valueChanged() - This method is called when the user selects any manyitem from the menubar.
Parameters:   lse - a ListSelectionEvent object containing details of the event.
Return Value: NA
*/
public void valueChanged(ListSelectionEvent lse)
{
```

```
        try
        {
           /* This section is executed, when end user selects the item from Font list */
           if(lse.getSource() == fontList)
           {
           Font f1 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex(),
           Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
           fontText.setText(String.valueOf(fontList.getSelectedValue()));
           label.setFont(f1);
           }
           /* This section is executed, when end user selects the item from Size list */
           else if(lse.getSource() == sizeList)
           {
              Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
              Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
              sizeText.setText(String.valueOf(sizeList.getSelectedValue()));
              label.setFont(f2);
           }
           /* This section is executed, when end user selects the item from Type list */
           else if(lse.getSource() == typeList)
           {
              Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
              Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
              typeText.setText(String.valueOf(typeList.getSelectedValue()));
              label.setFont(f2);
           }
        }
        catch(Exception nfe)
        {}
     }
     /*
     font() - This method is set the font of the file text
     Parameters:   NA
     Return Value: font
      */
     public Font font()
     {
        /* Create an object of Font class */
        Font font = new Font(String.valueOf(fontList.getSelectedValue()), typeList.getSelectedIndex(),
        Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
        /* Return the font object */
        return font;
     }
}
```

Download this listing.

The user interface of the FontChoose.java file provides the Selecting the Font dialog box. The end user can select the style, type, and size of the font from the Selecting the Font dialog box, as shown in Figure 4-7:

Figure 4-7: The Selecting the Font Dialog Box

Clicking OK or Cancel on the user interface of the Selecting the Font dialog box invokes the actionPerformed() method. The OK button sets the font of the text in the document to be printed to the selected font. The Cancel button cancels the current selection and the control returns to the user interface of the CommPrinting.java file.

PREVIOUS NEXT

Printing the Document

The PrintComponent_Class.java file allows an end user to specify setup properties for the page and print the text in the document that is currently open in the Printing application. The PrintComponent_Class file uses the classes of the java.awt.Print package to specify and set the properties of the document to be printed.

Listing 4-5 shows the PrintComponent_Class.java file:

Listing 4-5: The PrintComponent_Class.java File

```
import java.awt.*;
import java.awt.print.*;
import java.awt.geom.*;
import javax.swing.*;
class PrintComponent_Class implements Printable
{
    private Component component;
    PrinterJob printJob = PrinterJob.getPrinterJob();
    PageFormat pageFormat= printJob.defaultPage();

    public static void printComponent(Component c)
    {
        new PrintComponent_Class(c).print();
    }
    public PrintComponent_Class(Component component)
    {
        this.component= component;
    }
    /* Set up the page and print. */
    public void pageSetupAndPrint()
    {
        pageFormat = printJob.pageDialog(pageFormat);
        printJob.setPrintable(this, pageFormat);
        if (printJob.printDialog())
        {
            try
            {
                printJob.print();
            }
            catch(PrinterException pe)
            {
               System.out.println("Error in printing !!! " + pe);
            }
        }
    }
    /* Print the page. */
    public void print()
    {
        printJob.setPrintable(this, pageFormat);
        if (printJob.printDialog())
        {
            try
            {
                printJob.print();
            }
            catch(PrinterException pe)
            {
               System.out.println("Error in printing !!! " + pe);
            }
        }
    }
    public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
    {
      Graphics2D g2 = (Graphics2D)g;
      Dimension d = component.getSize(); //get size of document
      double componentWidth = d.width; //width in pixels
      double componentHeight = d.height; //height in pixels
      double pageHeight = pf.getImageableHeight(); //height of printer page
      double pageWidth = pf.getImageableWidth(); //width of printer page
      double scale = pageWidth/componentWidth;
      int pages = (int)Math.ceil(scale * componentHeight / pageHeight);
       /* Do not print empty pages. */
       if(pageIndex >= pages)
       {
            return Printable.NO_SUCH_PAGE;
       }
      /* Shift Graphic to line up with beginning of print-imageable region */
      g2.translate(pf.getImageableX(), pf.getImageableY());
      /* shift Graphic to line up with beginning of next page to print. */
      g2.translate(0f, -pageIndex*pageHeight);
      /* Scale the page so that the width fits. */
      g2.scale(scale, scale);
      /* Repaint the page. */
      component.paint(g2);
```

```
        return Printable.PAGE_EXISTS;
    }
}
```

Download this listing.

The above listing invokes the PrintComponent_Class.java file when the end user selects the Page Setup and Print menu option. The PrintComponent_Class.java file defines the print() and pageSetupAndPrint() methods to print and specify the page setup properties of the document to be printed. PrintComponent_Class.java uses the PageFormat and PrinterJob classes of Java to print the text in the document.

PREVIOUS NEXT

Unit Testing

To test the Printing application:

1. Download the Javacomm20-win32 JCA, which is available in zip format. This JCA is available at the following URL:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java 2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the CommPrinting.java, PortChoice.java, FontChoose.java, ColorChoose.java, and Printcomponent_Class.java files to a folder on your computer. At the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Run the Printing application using the following command at the command prompt:

   ```
   java CommPrinting
   ```

10. Select File->Open to open the text document that you want to print. The selected document opens in the user interface, as shown in Figure 4-8:

Figure 4-8: The Printing Application Interface

11. Click File->Choose COM Port to Print to select the port to which the printer is connected. You can select the port to use for printing from the list of ports that appears on the user interface of PortChoice.java and click OK.
12. Select Font->Font to set the font of the text in the document to be printed. The Selecting the Font dialog box appears.
13. Select the type, style, and size of the font of the text in the document and click OK. The font of the text changes to the selected font, as shown in Figure 4-9:

Figure 4-9: Changing the Font

14. Select Color->Color to select the color of the text in the document to be printed. The Selecting the Color dialog box opens. The end user can select a specific color for the text and click OK. This changes the color of the text in the document to the selected color, as shown in Figure 4-10:

Figure 4-10: Changing the Color

15. Select File->Page Setup and Print to print the text in the document that is currently opened. This opens the Page setup dialog box, as shown in Figure 4-11:

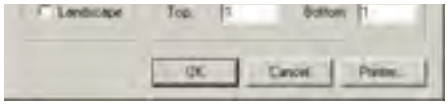

Figure 4-11: The Page Setup Dialog Box

16. Click OK. The Print dialog box opens, as shown in Figure 4-12:

Figure 4-12: The Print Dialog Box

17. Click OK.

◄ PREVIOUS NEXT ►

Chapter 5: Creating a Serial Communication Application

The Java Communication Application Programming Interface (JCA) supports the javax.comm package. This package enables you to use the CommPortIdentifier class to obtain a list of the COM or serial ports on a computer. The javax.comm package also provides the SerialPortEventListener and CommPortOwnershipListener interfaces, which include methods to handle and propagate events associated with serial or COM ports.

This chapter explains how to develop a Serial Communication application using the javax.comm package to send and receive data between the two COM or serial ports on two different computers that are attached using a null modem wire.

Architecture of the Serial Communication Application

The Serial Communication application enables two COM or serial ports to communicate with each other, which enables end users to communicate with each other. The ports of the two computers are connected through a null modem wire. The end user can use the Serial Communication application to specify the communication properties of the ports.

The Serial Communication application uses the following files:

- PortCommunication.java: Creates a user interface that helps an end user open a COM or serial port for communication, and send and receive data from the COM or serial port. This file invokes the PortPropertyPage.java file to display and set the properties of the port used for communication.
- PortPropertyPage.java: Enables the end user to specify the properties of a COM or serial port. The various properties of a COM or serial port that an end user can set are baud rate, stop bit, data bit, or parity bit.

Figure 5-1 shows the architecture of the Serial Communication application:

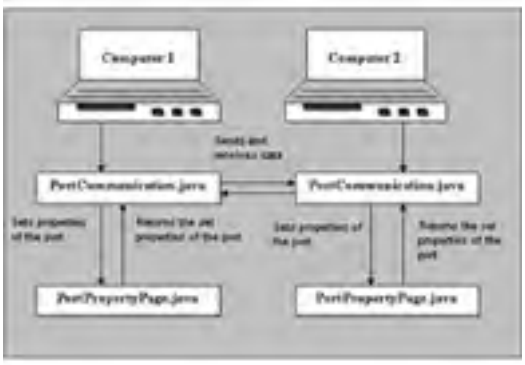

Figure 5-1: Architecture of the Serial Communication Application

Creating the User Interface and Implementing Business Logic

The PortCommunication.java file is the main file of the Serial Communication application. This file creates a user interface for the Serial Communication application. Listing 5-1 shows the PortCommunication.java file:

Listing 5-1: The PortCommunication.java File

```
import javax.comm.*;
/*Imports required java.i/o package classes.*/
import java.io.File;
import java.io.*;
/*Imports required Filedialogbox classes.*/
import javax.swing.filechooser.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Import the java.util package classes.*/
import java.util.*;
/*
class PortCommunication - This class is the main class of the application.
This class initializes the interface and loads all components like the menubar,
textareas, and buttons before displaying the result.
Constructor:
   PortCommunication-This constructor creates GUI.
Methods:
   createGUI - This method creates GUI.
   actionPerformed - This method describes which action will be performed on which component.
   getExtension - This method gives file extension.
   setConnectionParameters
   openPort - This method opens port.
   closePort - This method closes port.
   writeFile - This method write contents of textarea into file.
   save - This method save contents of textarea into file.
   main - This method creates the main window of the application and displays all the components.
*/
public class PortCommunication extends JFrame implements ActionListener,
SerialPortEventListener, CommPortOwnershipListener
{
   /*Declare object of JMenuBar class.*/
   private JMenuBar menubar;
   /* Declare object of JMenu class.*/
   private JMenu filemenu;
   /*Declare object of JMenuItem class.*/
   private JMenuItem openmenuitem, savemenuitem, exitmenuitem, property menuitem;
   /*Declare object of JTextArea class.*/
   private JTextArea sendertextarea;
   private JTextArea recevertextarea;
   /*Declare object of JScrollPane class.*/
   private JScrollPane senderscrollpane;
   private JScrollPane receverscrollpane;
   /*Declare object of JPanel class.*/
   private JPanel buttonpanel;
   private JPanel textareapanel;
   /*Declare object of JButton class.*/
   private JButton openportbutton;
   private JButton closeportbutton;
   /*Declare object of JComboBox class.*/
   private JComboBox comportcombo;
   /*
   Declare and Initialize object of File class.
   */
   File file=null;
   /*Declare object of JFileChooser class.*/
   JFileChooser openfile;
   /* Declare object of OutputStream class.*/
   private OutputStream os;
   /*Declare object of InputStream class.*/
   private InputStream is;
   /*Declare object of KeyHandler class.*/
   private KeyHandler keyHandler;
   /*Declare object of PortPropertyPage class.*/
   PortPropertyPage portpropertypage;
   /*
   Declare object of CommPortIdentifier class.
   */
   private CommPortIdentifier opencomm;
   /*Declare object of SerialPort class.*/
   private SerialPort sport;
   boolean open=false;
   private StringBuffer outputbuffer=new StringBuffer();
   public PortCommunication()
```

```
{
   super("Serial Communication");
   createGUI();
}
/*
createGUI: It is called to create menubar, text areas, and button.
   Parameter: N/A
   Return Value: N/A
*/
public void   createGUI()
{
   /*
   Set location at which window will be displayed.
   */
   Dimension  scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
   setLocation((scrnSize.width / 2) - 250, (scrnSize.height / 2) - 250);
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Error in setting WLAF"+e);
   }
   /* Set window's close operation. */
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   addWindowListener(new WindowAdapter()
   {
      public void windowOpened(WindowEvent we)
      {
         System.out.println("Window opened!");
      }
   });
   /*
   Declare and Initialize object of Container class with return value of getContentPane function.
   */
   Container contentpane=getContentPane();
   /*Initialize object of JFileChooser.*/
   openfile=new JFileChooser();
   /*Set file filter for openfile object.*/
   openfile.setFileFilter(new javax.swing.filechooser.FileFilter ()
   {
   /*
   accept: Override method of FileFilter class.
   Parameter: f - Object of File class.
   Return Value:boolean
   */
   public boolean accept(File f)
   {
      if (f.isDirectory())
      {
         return true;
      }
      String name = f.getName();
      if (name.endsWith(".txt"))
      {
         return true;
      }
      return false;
   }
   /*
   getDescription: Override method of FileFilter class.
   Parameter: NA
   Return Value: String
   */
         public String getDescription()
   {
      return ".txt";
         }
        });
   /*
   Initialize the object of JMenuBar class.
   */
   menubar=new JMenuBar();
   /*
   Initialize the object of JMenu class.
   */
   filemenu=new JMenu("File");
   filemenu.setMnemonic('f');
   /*
   Initialize the object of JMenuItem class.Set hot key as 's' and add to file menu.
   */
   savemenuitem=new JMenuItem("Save");
   savemenuitem.addActionListener(this);
   savemenuitem.setMnemonic('s');
```

```
savemenuitem.setActionCommand("save");
filemenu.add(savemenuitem);
/*
Initialize the object of JMenuItem class. Set hot key as 'o' and add to file menu.
*/
propertymenuitem=new JMenuItem("Properties");
propertymenuitem.addActionListener(this);
propertymenuitem.setActionCommand("open_property");
propertymenuitem.setMnemonic('o');
filemenu.add(propertymenuitem);
/*
Initialize the object of JMenuItem class.Set hot key as 'e' and add to file menu.
*/
exitmenuitem=new JMenuItem("Exit");
exitmenuitem.addActionListener(this);
exitmenuitem.setActionCommand("exit");
exitmenuitem.setMnemonic('e');
filemenu.add(exitmenuitem);
/*
Initialize the object of JMenu class.
*/
menubar.add(filemenu);
setJMenuBar(menubar);
/* Initialize object of JPanel.*/
buttonpanel=new JPanel();
/*
Initialize object of JButton, set title, hotkey and action command of this button
*/
openportbutton=new JButton("<html>O<u>p</u>en Port</html>");
openportbutton.setMnemonic('p');
openportbutton.setActionCommand("open_port");
openportbutton.addActionListener(this);
/*
Initialize object of JButton, set title, hotkey and action command of this button
*/
closeportbutton=new JButton("<html><u>C</u>lose Port</html>");
closeportbutton.setMnemonic('C');
closeportbutton.setEnabled(false);
closeportbutton.setForeground(Color.gray);
closeportbutton.setActionCommand("close_port");
closeportbutton.addActionListener(this);
/*
Declare and initialize objects of GridBagLayout class.
*/
GridBagLayout gridbaglayout=new GridBagLayout();
/*
Declare and initialize object of GridBagConstraints class.
*/
GridBagConstraints gridbagconstraint=new GridBagConstraints();
/*
Set buttonpanel's layout as gridbaglayout
*/
buttonpanel.setLayout(gridbaglayout);
/*
Set properties of gridbagconstraint
*/
gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(openportbutton, gridbagconstraint);
/*
Add button to buttonpanel pane
*/
buttonpanel.add(openportbutton);
/*
Set properties of gridbagconstraint
*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(closeportbutton, gridbagconstraint);
/*
Add button to buttonpanel pane
*/
buttonpanel.add(closeportbutton);
contentpane.add(buttonpanel,BorderLayout.SOUTH);
textareapanel=new JPanel(new GridLayout(2,1,5,5));
/*
Initialize objects of JTextArea class.
*/
sendertextarea=new JTextArea();
recevertextarea=new JTextArea();
/*
Set recevertextarea as non editable.
*/
```

```
        recevertextarea.setEditable(false);
        /*
        Initialize objects of JScrollPane class.
        */
        senderscrollpane=new JScrollPane(sendertextarea);
        receverscrollpane=new JScrollPane(recevertextarea);
        /*
        Add scrollPanes to textareapanel
        */
        textareapanel.add(senderscrollpane);
        textareapanel.add(receverscrollpane);
        contentpane.add(textareapanel,BorderLayout.CENTER);
        /*
        Initialize object of PortPropertyPage
        */
        portpropertypage=new PortPropertyPage();
        /*
        Add actionlistener with ok button of PortPropertyPage class.
        */
        portpropertypage.getOkButton().addActionListene r(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              if (open)
              {
                 System.out.println(portpropertypage.getCommPortName()+"--"+sport.getName());
                 if (portpropertypage.getCommPortName()==sport.getName())
                 {
                    setConnectionParameters();
                 }
                 else
                 {
                    System.out.println("Port can't be changed while another port is open");
                 }
              }
              else
              {
                 setConnectionParameters();
              }
              portpropertypage.setVisible(false);
           }
        });
        /*
        Add actionlistener with cancel button of PortPropertyPage class.
        */
        portpropertypage.getCancelButton().addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent ae)
           {
              portpropertypage.setVisible(false);
           }
        });
        /*
        Set size of main window.
        */
        setSize(400,300);
        setVisible(true);
     }
     /*
  Main method that creates the instance of the PortCommunication class.
     */
     public static void main(String args[])
     {
        PortCommunication portcommunication=new PortCommunication();
     }
     /*
     actionPerformed - This method is called when the user clicks the Open Port or
     Close Port button, selects file or property menu item.
     Parameters: ae - an ActionEvent object containing details of the event.
     Return Value: NA
      */
     public void actionPerformed(ActionEvent ae)
     {
     String componentid=ae.getActionCommand();
     /*
     This is executed when user clicks the Open port button.
     */
     if (componentid=="open_port")
     {
        if (openPort())
        {
           System.out.println("open");
           openportbutton.setEnabled(false);
           openportbutton.setForeground(Color.gray);
           closeportbutton.setEnabled(true);
           closeportbutton.setForeground(Color.black);
           System.out.println(open);
        }
```

```
    }
    /*
    This is executed when user clicks the Close port button.
    */
    if (componentid=="close_port")
    {
       if (closePort())
       {
          System.out.println("close");
          closeportbutton.setEnabled(false);
          openportbutton.setEnabled(true);
          closeportbutton.setForeground(Color.gray);
          openportbutton.setForeground(Color.black);
          System.out.println(open);
       }
    }
    /*
    This is executed when user clicks the Open property Page menu item.
    */
    if (componentid=="open_property")
    {
       portpropertypage.setVisible(true);
    }
    /*
    This is executed when user clicks the Save menu item.
    */
    if (componentid=="save")
    {
       save();
    }
    /*
    This is executed when user clicks the Exit menu Item.
    */
    if (componentid=="exit")
    {
       System.exit(0);
    }
}
    /*
closePort - This method is called to close an opened port.
Parameters: NA
Return Value:boolean
 */
public boolean closePort()
{
    if (!open)
    {
       return false ;
    }
    sendertextarea.removeKeyListener(keyHandler);
    if (sport != null)
    {
       try
       {
          /* Close the I/O streams.*/
          os.close();
          is.close();
       }
       catch (IOException e)
       {
          System.err.println(e);
       }
       /*Close serial port.*/
       sport.close();
       opencomm.removePortOwnershipListener(this);
    }
    open=false;
    return true;
}
    /*
    openPort - This method is called to open a closed port.
       Parameters: NA
       Return Value:boolean
    */
    public boolean openPort()
    {
       if (portpropertypage.getCommPortName()==null||portpropertypage.getCommPortName()=="")
       {
          JOptionPane.showMessageDialog(null," No port exists. "," Error ",JOptionPane.PLAIN_MESSAGE)
         return false;
       }
       opencomm=portpropertypage.getCommPort();
       try
       {
          sport=(SerialPort)opencomm.open("Serial Communication",2000);
          try
          {
             setConnectionParameters();
```

```
            }
            catch(Exception e)
            {
               JOptionPane.showMessageDialog(null,"  Properties can not be set for this port. ","
               Serial Communication ",JOptionPane.PLAIN_MESSAGE);
            }
         }
         catch(PortInUseException pe)
         {
            JOptionPane.showMessageDialog(null," Port is in use. "," Serial Communication ",JOptionPane
         }
         try
         {
            os = sport.getOutputStream();
            is = sport.getInputStream();
         }
         catch (IOException e)
         {
            sport.close();
            return false;
         }
         keyHandler = new KeyHandler(os);
         sendertextarea.addKeyListener(keyHandler);
         try
         {
            sport.addEventListener(this);
         }
         catch (TooManyListenersException e)
         {
            sport.close();
            return false;
         }
         sport.notifyOnDataAvailabel(true);
         sport.notifyOnBreakInterrupt(true);
         try
         {
            sport.enableReceiveTimeout(30);
         }
         catch (UnsupportedCommOperationException e)
         {
            JOptionPane.showMessageDialog(null,"Serial Communication","
            This operation is not supported by this port.",JOptionPane.PLAIN_MESSAGE);
            sport.close();
            return false;
         }
         opencomm.addPortOwnershipListener(this);
         open=true;
         return true;
      }
      /*
      setConnectionParameters - This method is called to set serial port properties.
      Parameters: NA
      Return Value:boolean
       */
      public void setConnectionParameters()
      {
         int oldbaudrate= 0;
         int olddatabits =0;
         int oldstopbits = 0;
         int oldparity = 0;
         int oldflowcontrol =0;
         if (open)
         {
             oldbaudrate = sport.getBaudRate();
             olddatabits = sport.getDataBits();
             oldstopbits = sport.getStopBits();
             oldparity = sport.getParity();
             oldflowcontrol = sport.getFlowControlMode();
            try
            {
               sport.setSerialPortParams(portpropertypage.getBaudRate(),
               portpropertypage.getDatabits(), portpropertypage.getStopbits(), _
               portpropertypage.getParity());
            }
            catch (UnsupportedCommOperationException e)
            {
               portpropertypage.setBaudRate(oldbaudrate);
               portpropertypage.setDatabits(olddatabits);
               portpropertypage.setStopbits(oldstopbits);
               portpropertypage.setParity(oldparity);
            }
            try
            {
               sport.setFlowControlMode(portpropertypage.getFlowControlIn() |
               portpropertypage.getFlowControlOut());
            }
            catch (UnsupportedCommOperationException ue)
            {
```

```
      }
   }
}
/*
getExtension: This method will extract file extension from file.
Parameter: f - object of File class
Return Value: file extension
*/
public String getExtension(File f)
{
   if(f != null)
   {
      String filename = f.getName();
      int i = filename.lastIndexOf('.');
      if(i>0 && i<filename.length()-1)
      {
         return filename.substring(i+1).toLowerCase();
      };
   }
   return null;
}
/*
save: This method will save to the contents of textarea into a file.
   Parameter- NA
   Return Value:boolean
*/
public boolean save()
{
   int result = openfile.showSaveDialog(this);
   if( result == JFileChooser.CANCEL_OPTION)
   {
      return true;
   }
   else if( result == JFileChooser.APPROVE_OPTION)
   {
      file = openfile.getSelectedFile();
      if( file.exists())
      {
         int response = JOptionPane.showConfirmDialog(null,
         "Overwrite existing file?",
         "Confirm Overwrite",
         JOptionPane.OK_CANCEL_OPTION,
         JOptionPane.QUESTION_MESSAGE);
         if( response == JOptionPane.CANCEL_OPTION)
            return false;
      }
      return writeFile(file, sendertextarea.getText());
   }
   else
   {
      return false;
   }
}
/*
writeFile: This method will save to the contents of textarea into a file.
   Parameter- file - object of File class, dataString - Object of String class.
   Return Value:boolean
*/
public static boolean writeFile(File file, String dataString)
{
   try
   {
      PrintWriter out=new PrintWriter(new BufferedWriter(new FileWriter(file)));
      out.print(dataString);
      out.flush();
      out.close();
   }
   catch (IOException e)
   {
      return false;
   }
   return true;
}
class KeyHandler extends KeyAdapter
{
   OutputStream os;
   public KeyHandler(OutputStream os)
   {
      super();
      this.os = os;
   }
   public void keyTyped(KeyEvent evt)
   {
         char newCharacter = evt.getKeyChar();
      try
      {
         if ((int)newCharacter==10)
         {
```

```
                for (int i=0;i<outputbuffer.length() ;i++ )
                {
                   os.write((int)outputbuffer.charAt(i));
                }
                outputbuffer=new StringBuffer();
             }
             else
             {
                outputbuffer.append(newCharacter);
             }
             System.out.println((int)newCharacter);
          }
          catch (IOException e)
          {
             System.err.println("OutputStream write error: " + e);
          }
       }
    }
    public void ownershipChange(int type)
    {
       if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED)
       {
       }
    }

    public void serialEvent(SerialPortEvent e)
    {
       /*
       Create a StringBuffer and int to receive input data.
       */
       StringBuffer inputBuffer = new StringBuffer();
       int newData = 0;
       /* Determine type of event.*/
       switch (e.getEventType())
       {
          case SerialPortEvent.DATA_AVAILABEL:
          while (newData != -1)
          {
             try
             {
                newData = is.read();
                if (newData == -1)
                {
                   break;
                }
                if ('\r' == (char)newData)
                {
                   inputBuffer.append('\n');
                }
                else
                {
                   inputBuffer.append((char)newData);
                }
             }
             catch (IOException ex)
             {
                System.err.println(ex);
                return;
             }
          }
          /* Append received data to messageAreaIn.*/
          recevertextarea.append(new String(inputBuffer));
             break;
          /*
          If break event append BREAK RECEIVED message.
          */
          case SerialPortEvent.BI:
             recevertextarea.append("\n--- BREAK RECEIVED ---\n");
       }
    }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortCommunication class. This class generates the main window of the Serial Communication application, as shown in Figure 5-2:

Figure 5-2: The Serial Communication Window

The user interface of the Serial Communication application displays a File menu that contains three menu options, Save, Properties, and Exit, as shown in Figure 5-3:

Figure 5-3: The File Menu

The Save menu option saves the data sent and received using the ports.

The Properties menu option enables the end user to set the properties of the port used for communication. When the end user clicks this option, it creates an instance of the PortPropertyPage.java class to open a dialog box that end users can use to change the properties of the selected port.

The Exit menu option enables the end user to exit from the application.

The user interface of the Serial Communication application also contains two text panes. The upper text pane sends data through a selected and opened port. The lower pane displays the data received using the selected and opened port.

In addition to text panes and menus, the user interface of the Serial Communication application also displays two buttons, Open Port and Close Port, to open and close the port used for communication.

When an end user clicks any button, the Serial Communication application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate method based on the button that the end user clicks. For example, when the end user clicks the Open Port button, the actionPerformed() method opens a port for communication. The actionPerformed() method invokes the setConnectionParameters() method, which returns the properties last saved by the end user. When the end user clicks the Close Port button, the actionPerformed() method closes the port that was opened for communication.

◄ PREVIOUS NEXT ►

Displaying Port Properties

The PortPropertyPage.java file allows the end user to display and set the properties of the port used to send and receive data. The various properties of the port that the PortPropertyPage.java file displays are port name, baud rate, flow control in, flow control out, data bits, stop bits, and parity bits.

Listing 5-2 shows the PortPropertyPage.java file:

Listing 5-2: The PortPropertyPage.java File

```
/*Imports required Comm classes*/
import javax.comm.*;
/*Imports required javax.swing package classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Imports required java.util package classes.*/
import java.util.*;
/*
class PortPropertyPage - This class is the Port Property dialog window,
Enable the end user to change the property of port.
Constructor:
   PortPropertyPage-This constructor creates GUI.
Methods:
   setParity- Sets Parity bit.
   setStopbits - Sets Stop bit.
   setDatabits - Set Data bit
   setBaudRate - Sets Baud rate.
   getCancelButton - Returns object of cancel button.
   getOkButton -  Returns object of ok button.
   getFlowControlOut - Returns flow control out value.
   getFlowControlIn - Returns flow control in value.
   getParity - Returns Parity bit value.
   getDatabits - Returns Data bit value.
   getStopbits - Returns Stop bit value.
   getBaudRate - Returns Baud rate value.
   getCommPortName - Returns  port name.
   getCommPort -  Returns object of CommPort class
*/
class PortPropertyPage extends JFrame   implements ActionListener
{
   /*Declare object of Enumeration class.*/
   Enumeration listport;
   /*Declare object of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /*Declare object of JPanel class.*/
   JPanel buttonpanel;
   JPanel propertylist;
   /*Declare object of JLabel class.*/
   JLabel portlistlabel;
   JLabel flowlabel;
   JLabel paritylabel;
   JLabel buadlabel;
   JLabel databitlabel;
   JLabel flowcontrollabel;
   JLabel stoplabel;
   /*Declare object of JComboBox class.*/
   JComboBox comportcombo;
   JComboBox flowcombo;
   JComboBox stopcombo;
   JComboBox databitcombo;
   JComboBox paritycombo;
   JComboBox buadcombo;
   JComboBox flowcontrolcombo;
   /*
   Declare object of CommPortIdentifier class.
   */
   CommPortIdentifier portId;
   public PortPropertyPage()
   {
      /*Set title of main window.*/
      super("Port Property Page");
      /*
      Declare and initialize object of Container class.
      */
      Container contentpane=getContentPane();
      comportcombo=new JComboBox();
      listport=CommPortIdentifier.getPortIdentifiers();
      while (listport.hasMoreElements())
      {
         portId=(CommPortIdentifier)listport.nextElement();
```

```
   if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
   {
      comportcombo.addItem(portId.getName());
   }
 }
/*Initialize Object of JComboBox.*/
flowcombo=new JComboBox();
flowcombo.addActionListener(this);
/*
Declare and initialize object of GridBagLayout class
*/
GridBagLayout gridbaglayout=new GridBagLayout();
/*
Declare and initialize object of GridBagConstraints class
*/
GridBagConstraints gridbagconstraint=new GridBagConstraints();
/*
Initialize object of JPanel class and set its layout as gridbaglayout.
*/
propertylist=new JPanel(gridbaglayout);
/*Set background color as white.*/
propertylist.setBackground(Color.white);
/*
Initialize object of JLabel class and gives its name.
*/
portlistlabel=new JLabel("Port Name",JLabel.RIGHT);
/*
Initialize object of JLabel class and gives its name.
*/
flowlabel=new JLabel("Flow Control In");
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=0;
gridbagconstraint.weightx=1.0;
gridbagconstraint.weighty=1.0;
gridbaglayout.setConstraints(portlistlabel, gridbagconstraint);
/*Add portlistlabel label to panel.*/
propertylist.add(portlistlabel);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbagconstraint.insets=new Insets(5, 5, 5, 5);
gridbaglayout.setConstraints(comportcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(comportcombo);
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowlabel, gridbagconstraint);
/*Add flowlabel label to panel.*/
propertylist.add(flowlabel);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(flowcombo);
/*
Initialize object of JLabel class and gives its name.
*/
databitlabel=new JLabel("Data Bits",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstrain ts.CENTER;
gridbaglayout.setConstraints(databitlabel, gridbagconstraint);
/*Add databitlabel label to panel.*/
propertylist.add(databitlabel);
/*Initialize object of JComboBox class.*/
databitcombo=new JComboBox();
databitcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(databitcombo, gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(databitcombo);
/*
Initialize object of JLabel class and gives its name.
```

```
*/
paritylabel=new JLabel("Parity Bits", JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=0;
gridbagconstraint.gridy=3;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(paritylabel, gridbagconstraint);
/*Add paritylabel label to panel.*/
propertylist.add(paritylabel);
/*Initialize object of JComboBox class.*/
paritycombo=new JComboBox();
paritycombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=1;
gridbagconstraint.gridy=3;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(paritycombo, gridbagconstraint);
/*Add paritycombo combo to panel.*/
propertylist.add(paritycombo);
/*
Initialize object of JLabel class and gives its name.
*/
buadlabel=new JLabel("Baud Rate", JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(buadlabel, gridbagconstraint);
/*Add buadlabel label to panel.*/
propertylist.add(buadlabel);
/*Initialize object of JComboBox class.*/
buadcombo=new JComboBox();
buadcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(buadcombo, gridbagconstraint);
/*Add buadcombo combo to panel.*/
propertylist.add(buadcombo);
/*
Initialize object of JLabel class and gives its name.
*/
flowcontrollabel=new JLabel("Flow Control Out",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrollabel, gridbagconstraint);
/*Add flowcontrollabel label to panel.*/
propertylist.add(flowcontrollabel);
/*Initialize object of JComboBox class.*/
flowcontrolcombo=new JComboBox();
flowcontrolcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrolcombo, gridbagconstraint);
/*Add flowcontrolcombo combo to panel.*/
propertylist.add(flowcontrolcombo);
/*
Initialize object of JLabel class and gives its name.
*/
stoplabel=new JLabel("Stop Bit",JLabel.RIGHT);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(stoplabel, gridbagconstraint);
/*Add stoplabel label to panel.*/
propertylist.add(stoplabel);
/*Initialize object of JComboBox class.*/
stopcombo=new JComboBox();
stopcombo.addActionListener(this);
/*Set constraints for gridbagconstraint.*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(stopcombo, gridbagconstraint);
/*Add stopcombo combo to panel.*/
propertylist.add(stopcombo);
contentpane.add(propertylist);
/* Fill buadcombo combobox. */
buadcombo.addItem("300");
buadcombo.addItem("2400");
buadcombo.addItem("9600");
buadcombo.addItem("14400");
```

```
   buadcombo.addItem("28800");
   buadcombo.addItem("38400");
   buadcombo.addItem("57600");
   buadcombo.addItem("152000");
   buadcombo.setSelectedItem("9600");
   /*Fill flowcombo combobox.*/
   flowcombo.addItem("None");
   flowcombo.addItem("Xon/Xoff In");
   flowcombo.addItem("RTS/CTS In");
   /*Fill flowcontrolcombo combobox.*/
   flowcontrolcombo.addItem("None");
   flowcontrolcombo.addItem("Xon/Xoff Out");
   flowcontrolcombo.addItem("RTS/CTS Out");
   /*Fill stopcombo combobox.*/
   stopcombo.addItem("1");
   stopcombo.addItem("1.5");
   stopcombo.addItem("2");
   /*Fill paritycombo combobox.*/
   paritycombo.addItem("None");
   paritycombo.addItem("Even");
   paritycombo.addItem("Odd");
   /*Fill databitcombo combobox.*/
   databitcombo.addItem("5");
   databitcombo.addItem("6");
   databitcombo.addItem("7");
   databitcombo.addItem("8");
   databitcombo.setSelectedItem("8");
   /*
   Initialize object of JButton and set its label
   */
   okbutton=new JButton("OK");
   /*
   Initialize object of JButton and set its label
   */
   cancelbutton=new JButton("Cancel");
   /*Initialize object of JPanel.*/
   buttonpanel=new JPanel();
   /*
   Set buttonpanel  layout as gridbaglayout.
   */
   buttonpanel.setLayout(gridbaglayout);
   /*Set constraints for gridbagconstraint.*/
   gridbagconstraint.fill = GridBagConstraints.HORIZONTAL;
   gridbagconstraint.insets=new Insets(5, 75, 5, 5);
   gridbagconstraint.gridx=0;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(okbutton, gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(okbutton);
   /*Set constraints for gridbagconstraint.*/
   gridbagconstraint.insets=new Insets(5, 5, 5, 75);
   gridbagconstraint.gridx=1;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(cancelbutton, gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(cancelbutton);
   contentpane.add(buttonpanel,BorderLayout.SOUTH);
   addWindowListener(new WindowAdapter()
   {
      public void windowClosed(WindowEvent we)
      {
         setVisible(false);
      }
   });
   /*Set frame size.*/
   setSize(400,200);
}
public void actionPerformed(ActionEvent e)
{
   JComboBox cb = (JComboBox)e.getSource();
   String petName = (String)cb.getSelectedItem();
}
/*
getCommPort - Gives object of  CommPort class
Parameters: NA
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getCommPort()
{
   CommPortIdentifier comp=null;
   try
   {
      comp= javax.comm.CommPortIdentifier.getPortIdentifier((String)comportcombo.getSelectedItem(
   }
   catch(NoSuchPortException e)
```

```
    {
       JOptionPane.showMessageDialog(null,"open", "Port not exists", JOptionPane.PLAIN_MESSAGE);
    }
    return comp;
}
/*
getCommPortName - Gives name of selected port.
Parameters: NA
Return Value:String
*/
public String getCommPortName()
{
    return (String)comportcombo.getSelectedItem();
}
/*
getBaudRate - Gives baud rate.
Parameters: NA
Return Value:int
*/
public int getBaudRate()
{
    return Integer.parseInt((String)buadcombo.getSelectedItem());
}
/*
getDatabits - Gives Databits.
Parameters: NA
Return Value:int
*/
public int getDatabits()
{
    return Integer.parseInt((String)databitcombo.getSelectedItem());
}
/*
getStopbits - Gives Stop bit.
Parameters: NA
Return Value:int
*/
public int getStopbits()
{
    return Integer.parseInt((String)stopcombo.getSelectedItem());
}
/*
    getParity - Gives parity bit.
    Parameters: NA
    Return Value:int
*/
public int getParity()
{
    String paritybit=(String)paritycombo.getSelectedItem();
    if (paritybit.equals("None"))
    {
       return SerialPort.PARITY_NONE;
    }
    else if (paritybit.equals("Even"))
    {
       return SerialPort.PARITY_EVEN;
    }
    else if (paritybit.equals("Odd"))
    {
        return SerialPort.PARITY_ODD;
    }
    else
    {
          return SerialPort.PARITY_NONE;
    }
}
/*
getFlowControlIn - Gives type of flow control.
    Parameters: NA
    Return Value:int
*/
public int getFlowControlIn()
{
    String flowcomboin=(String)flowcombo.getSelectedItem();
    if (flowcomboin.equals("None"))
    {
       return SerialPort.FLOWCONTROL_NONE;
    }
    if (flowcomboin.equals("Xon/Xoff Out"))
    {
       return SerialPort.FLOWCONTROL_XONXOFF_OUT;
    }
    if (flowcomboin.equals("Xon/Xoff In"))
    {
       return SerialPort.FLOWCONTROL_XONXOFF_IN;
    }
    if (flowcomboin.equals("RTS/CTS In"))
    {
```

```
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowcomboin.equals("RTS/CTS Out"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }
    /*
    getFlowControlOut - Gives type of flow out control.
    Parameters: NA
    Return Value:int
    */
    public int getFlowControlOut()
    {
        String flowcontrol=(String)flowcontrolcombo.getSelectedItem();
        if (flowcontrol.equals("None"))
        {
            return SerialPort.FLOWCONTROL_NONE;
        }
        if (flowcontrol.equals("Xon/Xoff Out"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_OUT;
        }
        if (flowcontrol.equals("Xon/Xoff In"))
        {
            return SerialPort.FLOWCONTROL_XONXOFF_IN;
        }
        if (flowcontrol.equals("RTS/CTS In"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowcontrol.equals("RTS/CTS Out"))
        {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }
    /*
    getOkButton - Gives object of JButton class
    Parameters: NA
    Return Value: JButton
    */
    public JButton getOkButton()
    {
        return okbutton;
    }

    /*
    getCancelButton - Gives object of JButton class
    Parameters: NA
    Return Value: JButton
    */
    public JButton getCancelButton()
    {
        return cancelbutton;
    }
    /*
        setBaudRate - Sets value of baud rate.
        Parameters: budrate
        Return Value: NA
    */
    public void setBaudRate(int budrate)
    {
        buadcombo.setSelectedItem(String.valueOf(budrate));
    }
    /*
    setDatabits - Sets value of Data bit.
    Parameters: databit
    Return Value: NA
    */
    public void  setDatabits(int databit)
    {
    databitcombo.setSelectedItem(String.valueOf(databit));
    }
    /*
        setStopbits - Sets value of Stop bit.
        Parameters: - stopbit
        Return Value: NA
    */
    public void  setStopbits(int stopbit)
    {
    stopcombo.setSelectedItem(String.valueOf(stopbit));
    }
    /*
        setParity - Sets value of Parity bit.
        Parameters: parity
        Return Value: NA
```

```
   */
   public void  setParity(int parity)
   {
      switch(parity)
      {
         case SerialPort.PARITY_NONE:
         stopcombo.setSelectedItem("None");
         break;
         case SerialPort.PARITY_EVEN:
         stopcombo.setSelectedItem("Even");
         break;
         case SerialPort.PARITY_ODD:
         stopcombo.setSelectedItem("Odd");
         break;
         default:
         stopcombo.setSelectedItem("None");
      }
   }
}
```

Download this listing.

The above listing enables the end user to set the properties of the port. Figure 5-4 shows the interface of the PortPropertyPage.java file:

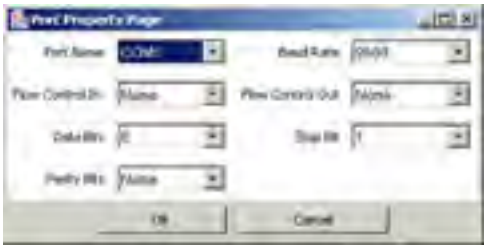

Figure 5-4: The Interface of the PortPropertyPage.java File

The user interface of the PortPropertyPage.java file displays various properties of the port. The PortPropertyPage.java file retrieves and specifies the properties of a port using the following get and set methods:

- The setParity() and getParity() methods of the PortPropertyPage class specify and retrieve the parity bit value of the port.
- The setStopbits() and getStopbits() methods of the PortPropertyPage class specify and retrieve the value of the stop bit of the port.
- The setBaudRate() and getBaudRate() methods of the PortPropertyPage class set and retrieve the baud rate property of the COM port.
- The setDatabits() and getDatabits() methods of the PortPropertyPage class specify and retrieve the value of the data bit property of the port.
- The getCancelButton() and getOKButton() methods of the PortPropertyPage class create the objects of the Cancel and OK buttons.
- The getFlowControlOut() and getFlowControlIn() methods of the PortPropertyPage class retrieve the out and in values of the flow control property.
- The getCommPorts() method of the PortPropertyPage class retrieves all the COM ports on a computer. The getCommPortName() method returns the name of the COM ports on a computer.

Unit Testing

To test the Serial Communication application:

1. Download the Javacomm20-win32 JCA, which is availabel in zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where Java 2 Software Development Kit (J2SDK) is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```

8. Copy the PortCommunication.java and PortPropertyPage.java files to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:

   ```
   javac *.java
   ```

9. Copy the PortCommunication.java and PortPropertyPage.java files to another computer on the network. Compile these files using the command specified in Step 8.
10. Connect the two computers that contain the PortCommunication.java and PortPropertyPage.java files using a null modem wire.
11. Run the Serial Communication application on both the computers connected by a null modem wire using the following command at the command prompt:

    ```
    java PortCommunication
    ```

12. Click the Open Port button in the Serial Communication window of both computers. This opens the currently selected port in the Port Property Page window for communication.
13. Click File->Properties in the Serial Communication window on both the computers connected by a null modem wire. This opens the Port Property page.
14. Select the port connected by the null modem wire in the Port Property Page window on both computers. You can set communication properties for the communicating ports in the Port Property Page window.
15. Click the OK button in the Port Property Page window to set the communication properties.
16. Type the data to be sent using the Serial Communication application. The data is specified in the upper text pane of the user interface of the Serial Communication window. Press Enter, as shown in Figure 5-5:

Figure 5-5: Sending Data Using the Serial Communication Application

The data is received in the lower pane of the Serial Communication interface on the other computer, as shown in Figure 5-6:

Figure 5-6: Receiving Data using the Serial Communication Application

PREVIOUS NEXT

Chapter 6: Creating a Fax Application

The javax.comm package includes the CommPortIdentifier class to control access to the various communication ports on a computer and the CommPort and ParallelPort classes to retrieve and set the properties of the ports on a computer that communicate with the modem and fax machine.

This chapter explains how to develop a Fax application that uses the Java Communication Application Programming Interface and the CommPort, ParallelPort, and CommPortIdentifier classes.

Architecture of the Fax Application

The Fax application allows an end user to specify a fax number, select a file, and send the content of the file as a fax to the specified number. The application sends the fax by converting a text file to an image. This process is called encoding the fax.

The Fax application uses the following files:

- PortFax.java: Creates the user interface of the Fax application.
- ConvertIntoHTML.java: Converts the text that is currently open in the edit pane of the Fax application to an image of an HTML page.
- ConvertIntoTextImage.java: Converts the text that is currently open in the edit pane of the Fax application to a text image.
- FaxPropertyPage.java: Sets the properties of the COM port that sends the fax.
- FaxStatusListener.java: Defines various static variables, such as ST_OPEN_PORT, ST_INIT_MODEM, and ST_SEND_PAGE, which indicate the status of the Fax application.
- SendingFax.java: Sends the fax to the destination.
- PreparedFaxDoc.java: Defines an interface to create the fax document. The ConvertIntoHTML.java and ConvertIntoTextImage.java files implement this interface.
- ModemCapabilities.java: Sets the encoding and decoding capabilities of a modem.
- ImageToFaxEncoder.java: Encodes the image that is sent using the Fax application.
- LCFrame.java: Converts the encoded fax image to frames. The modem sends the frames to the receiver.

Figure 6-1 shows the architecture of the Fax application:

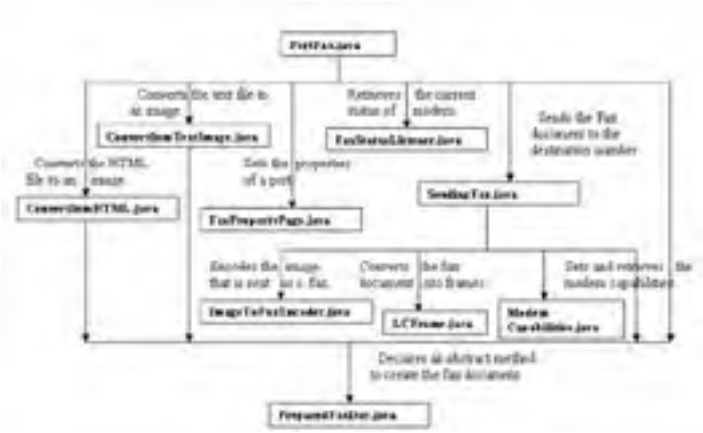

Figure 6-1: Architecture of the Fax Application

The PortFax.java file calls the ConvertIntoTextImage.java and ConvertIntoHTML.java files to convert the fax document to an image. The PortFax.java file invokes the FaxPropertyPage.java file to set the properties of the port to which the modem is connected. The FaxStatusListener.java file helps the PortFax.java file determine the status of the Fax application.

The end user sends a fax using the SendingFax.java file, which invokes the ImageToFaxEncoder.java file to encode the fax document. The LCFrame.java file helps the SendingFax.java file convert the encoded fax document into frames. The SendingFax.java file invokes the ModemCapabilities.java file to retrieve and set the encoding and decoding capabilities of a modem that helps the modem detect incoming frames to send a fax.

The PortFax.java file passes the frames to the modem, which in turn passes the frames to the fax number of the destination.

Creating the User Interface

The PortFax.java file is the main file of the Fax application. This file creates the user interface of the Fax application. The user interface of the Fax application consists of text fields to set the number to which the end user wants to send the fax, and the initial command string for the modem. In addition, the user interface of the Fax application contains an edit pane that displays the text file that the end user wants to send as a fax.

Listing 6-1 shows the PortFax.java file:

Listing 6-1: The PortFax.java File

```
/* Imports required javax.comm package classes */
import javax.comm.*;
/* Imports required java.i/o package classes */
import java.io.*;
/*Imports required java.awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required javax.swing package classes*/
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
/*
class PortFax - This class is the main class of the application. This class initializes the
interface and loads all components like the menubar, textareas, and buttons
before displaying the result.
Constructor:
   PortFax - This constructor creates GUI.
Methods:
   createGUI - This method creates GUI.
actionPerformed - This method describes which action will be performed on which component.
main - This method creates the main window of the application and displays all the components.*/

class PortFax extends JFrame implements ActionListener
{
   /*Set constants*/
   public static final int ST_OPEN_PORT = 1;
    public static final int ST_INIT_MODEM = 2;
    public static final int ST_CONNECTING = 3;
    public static final int ST_SEND_PAGE = 4;
    public static final int ST_CLOSE = 5;
    public static final int ST_CONVERT_FIlES = 6;
    public static final int ST_REC_PAGE = 4;
    public static final int ST_REC_CALL = 7;
    public static final int ST_WAIT_CALL = 8;
   /*Declare objects of String class.*/
   String oldcommname;
   String olddialing;
   String oldmodemclass;
   String oldflowcontrol;
   /*Declare objects of JLabel class.*/
   JLabel fexLabel;
   JLabel modemstringLabel;
   JLabel processlabel;
   /*Declare objects of JTextField class.*/
   JTextField modemstringtext;
   JTextField faxtextfield;
   /*Declare objects of JEditorPane class.*/
   JEditorPane faxeditor;
   /*Declare objects of JButton class.*/
   JButton sendbutton;
   JButton cancelbutton;
   JButton closebutton;
   /*Declare objects of JPanel class.*/
   JPanel toppanel;
   JPanel EditorPanel;
   /*Declare objects of JMenuBar class.*/
   JMenuBar menubar;
   /*Declare objects of JMenu class.*/
   JMenu filemenu;
   /*Declare objects of JMenuItem class.*/
   JMenuItem openmenuitem,viewpropertiesmenuitem,exitmenuitem;
   /*Declare objects of JFileChooser class.*/
   JFileChooser filechooser;
   /*Declare objects of JScrollPane class.*/
   JScrollPane editorscrollpane;
   /*Declare objects of JCheckBox class.*/
   JCheckBox htmlcheckbox;
   /*Declare objects of PreparedFaxDoc class.*/
   PreparedFaxDoc faxdoc=null;
   /*Declare objects of SendingFax class.*/
   SendingFax sendfax=null;
   /*Declare objects of FaxPropertyPage class.*/
```

```
FaxPropertyPage faxpropertypage;
int oldflowcontrolin=0;
boolean d=false;
public PortFax()
{
   /*Set parent window title.*/
   super(" Fax Application ");
   /*createGUI method is called*/
   createGUI();
   /*Set window size.*/
   setSize(500,500);
   /*Set window visible.*/
   setVisible(true);
}
public static void main(String[] args)
{
   /*Declare and initialize object of PortFax.*/
   PortFax portfax=new PortFax();
}
public void createGUI()
{
   /*
   Declare and Initialize object of Container class with return value of getContentPane function.
   */
   Container contentpane=getContentPane();
   /*
   Initialize and set the Look and Feel of the application to Windows Look and Feel.
   */
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      System.out.println("Problem in changing windows look and feel");
   }
   /*Set location at which window will be displayed.*/
   Dimension screensize=Toolkit.getDefaultToolkit()getScreenSize();
   setLocation(screensize.width/2-250,screensize.height/250);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   /*Initialize the object of JMenuBar class.*/
   menubar=new JMenuBar();
   /*Initialize the object of JMenu class.*/
   filemenu=new JMenu("File");
   filemenu.setMnemonic('f');
   /*
   Initialize the object of JMenuItem class.
   */
   openmenuitem=new JMenuItem("Open");
   openmenuitem.setMnemonic('o');
   openmenuitem.setActionCommand("open");
   openmenuitem.addActionListener(this);
   filemenu.add(openmenuitem);
   /*
   Initialize the object of JMenuItem class.
   */
   viewpropertiesmenuitem=new JMenuItem("View Properties");
   viewpropertiesmenuitem.setMnemonic('v');
   viewpropertiesmenuitem.setActionCommand("view");
   viewpropertiesmenuitem.addActionListener(this);
   filemenu.add(viewpropertiesmenuitem);
   /*
   Initialize the object of JMenuItem class.
   */
   exitmenuitem=new JMenuItem("Exit");
   exitmenuitem.setMnemonic('e');
   exitmenuitem.setActionCommand("exit");
   exitmenuitem.addActionListener(this);
   filemenu.add(exitmenuitem);
   menubar.add(filemenu);
   setJMenuBar(menubar);
   /*
   Initialize the object of FaxPropertyPage class.
   */
   faxpropertypage=new FaxPropertyPage();
   /*
   Add action listener to cancel button of faxpropertypage object.
   */
   faxpropertypage.getCancelButton()addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
      if (oldcommname!=null)
      {
         faxpropertypage.setCommPort(oldcommname);
      }
      else if(oldcommname.length()>0)
      {
```

```
            faxpropertypage.setCommPort(oldcommname);
        }
        if (olddialing!=null)
        {
            faxpropertypage.setdialingMode(olddialing);
        }
        else if (olddialing.length()>0)
        {
            faxpropertypage.setdialingMode(olddialing);
        }
        if(oldmodemclass!=null)
        {
            faxpropertypage.setModemClass(oldmodemclass);
        }
        else if(oldmodemclass.length()>0)
        {
            faxpropertypage.setModemClass(oldmodemclass);
        }
        if(oldflowcontrol!=null)
        {
            faxpropertypage.setFlowControl(oldflowcontrol);
        }
        else if(oldflowcontrol.length()>0)
        {
            faxpropertypage.setFlowControl(oldflowcontrol);
        }
        faxpropertypage.setFlowControlIn(oldflowcontrolin);
    }
    });
    /*Initialize object of JFileChooser.*/
    filechooser=new JFileChooser();
    /*
    Declare and initialize object of GridBagLayout class
    */
    GridBagLayout gridbaglayout=new GridBagLayout();
    /*
    Declare and initialize object of GridBagConstraints class
    */
    GridBagConstraints gridbagconstraints =new GridBagConstraints();
    /*
    Initialize object of JPanel and set its layout as GridBagLayout.
    */
    toppanel=new JPanel(gridbaglayout);
    /*
    Initialize object of JLabel and set constraints to this label
    */
    modemstringLabel=new JLabel("Modem Initial String");
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(10, 10, 5, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemstringLabel, gridbagconstraints);
    /*Add label to panel*/
    toppanel.add(modemstringLabel);
    /*
    Initialize object of JTextField and set constraints to this label
    */
    modemstringtext=new JTextField("ATV1Q0");
    modemstringtext.setPreferredSize(new Dimension(150, 23));
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(10, 10, 5, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemstringtext, gridbagconstraints);
    /*Add button to panel*/
    toppanel.add(modemstringtext);
    /*
    Initialize object of JLabel and set constraints to this label
    */
    fexLabel=new JLabel("Fax Number");
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=1;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(0, 10, 10, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(fexLabel, gridbagconstraints);
    /*Add label to panel*/
    toppanel.add(fexLabel);
    /*
    Initialize object of JTextField and set constraints to this label
    */
    faxtextfield=new JTextField("0, 51610236");
    faxtextfield.setPreferredSize(new Dimension(150, 23));
```

```
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=1;
    gridbagconstraints.weightx=1.0;
    gridbagconstraints.insets=new Insets(0, 10, 10, 10);
    gridbagconstraints.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(faxtextfield,gridbagconstraints);
    /*Add button to panel*/
    toppanel.add(faxtextfield);
    /*
    Initialize object of JCheckBox and set constraints to this label
    */
    htmlcheckbox=new JCheckBox(" HTML");
    htmlcheckbox.setBorderPaintedFlat(true);
    gridbagconstraints.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraints.gridx=2;
    gridbagconstraints.gridy=2;
    gridbagconstraints.weightx=0.0;
    gridbagconstraints.gridwidth=2;
    gridbagconstraints.insets=new Insets(0, 10, 0, 0);
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(htmlcheckbox, gridbagconstraints);
    /*Add checkbox to panel*/
    toppanel.add(htmlcheckbox);
    /*Add panel to main window*/
    contentpane.add(toppanel, BorderLayout.NORTH);
    /*Initialize object of editorpane.*/
    faxeditor=new JEditorPane();
    /*Initialize object of scrollpane.*/
    editorscrollpane=new JScrollPane(faxeditor);
    /*Add scrollbar to main window.*/
    contentpane.add(editorscrollpane,BorderLayout.CENTER);
    /*
    Declare and initialize object of JPanel class and set its layout as GridLayout.
    */
    JPanel bottomlabelpanel=new JPanel(new GridLayout(1, 1, 0, 0));
    bottomlabelpanel.add(processlabel=new JLabel(""));
    processlabel.setForeground(Color.blue);
    /*
     and initialize object of JPanel class and set its layout as GridLayout.
    */Declare
    JPanel bottombuttonpanel=new JPanel(new GridLayout(1, 5, 10, 10));
    bottombuttonpanel.add(new JLabel(""));
    /*Initialize object of JButton.*/
    sendbutton=new JButton(" Send Fax ");
    /*Set button size.*/
    sendbutton.setPreferredSize(new Dimension(20,23));
    sendbutton.setMnemonic('s');
    sendbutton.setActionCommand("send");
    sendbutton.addActionListener(this);
    bottombuttonpanel.add(sendbutton);
    cancelbutton=new JButton(" Cancel ");
    /*Set button size.*/
    cancelbutton.setPreferredSize(new Dimension(20,23));
    cancelbutton.setMnemonic('c');
    cancelbutton.setActionCommand("cancel");
    cancelbutton.addActionListener(this);
    bottombuttonpanel.add(cancelbutton);
    closebutton=new JButton(" Close ");
    /*Set button size.*/
    closebutton.setPreferredSize(new Dimension(20, 23));
    closebutton.setMnemonic('l');
    closebutton.setActionCommand("close");
    closebutton.addActionListener(this);
    bottombuttonpanel.add(closebutton);
    bottombuttonpanel.add(new JLabel(""));
    /*
    Declare and initialize object of JPanel class and set its layout as GridLayout.
    */
    JPanel bottompanel=new JPanel(new GridLayout(2, 1));
    bottompanel.add(bottomlabelpanel);
    bottompanel.add(bottombuttonpanel);
    /*Set panel size.*/
    bottompanel.setPreferredSize(new Dimension(550, 50));
    /*Add panel to main window.*/
    contentpane.add(bottompanel,BorderLayout.SOUTH);
}
/*
actionPerformed - This method is called when the user clicks the Send or Close button,
selects file menu items.
Parameters: ae - an ActionEvent object containing details of the event.
Return Value: NA
 */
public void actionPerformed(ActionEvent ae)
{
    String actioncommand=ae.getActionCommand();
    /*
    This is executed when user clicks the Open menu item.
```

```
*/
if ("open".equals(actioncommand))
{
   int returnVal =filechooser.showOpenDialog(PortFax.this);
   if (returnVal==JFileChooser.APPROVE_OPTION)
    {
      try
      {
         File file = filechooser.getSelectedFile();
         InputStream inputStream = new FileInputStream(file);
         BufferedReader br = new BufferedReader(new InputStreamReader(inputStream));
         String strTemp="";
         faxeditor.setText("");
         while ((strTemp = br.readLine())!=null)
         {
            faxeditor.setText(faxeditor.getText()+strTemp+'\n');
         }
         inputStream.close();
         }
         catch( FileNotFoundException e )
         {
            System.out.println( "File not found" );
         }
         catch( IOException e )
         {
            System.out.println( "I/O error" );
         }
       }
}
/*
This is executed when user clicks the View Property menu item.
*/
else if("view".equals(actioncommand))
{
   oldcommname=faxpropertypage.getCommPortName();
   olddialing=faxpropertypage.getDialing();
   oldmodemclass=(String)faxpropertypage.modemclasscombo.getSelectedItem();
   oldflowcontrol=faxpropertypage.getFlowcontrol();
   oldflowcontrolin=faxpropertypage.getFlowControlIn();
   faxpropertypage.setVisible(true);
}
/*
This is executed when user clicks the Exit menu item.
*/
else if("exit".equals(actioncommand))
{
   System.exit(0);
}
/*
This is executed when user clicks the Send button.
*/
else if ("send".equals(actioncommand))
{
   sendbutton.setEnabled(false);
   closebutton.setEnabled(true);
   if (htmlcheckbox.isSelected())
   {
      faxdoc= new ConvertIntoHTML();
      String[] pag=new String[1];
      pag[0]=faxeditor.getText();
      ((ConvertIntoHTML) faxdoc).text=pag;
      ((ConvertIntoHTML) faxdoc).pageimage=createImage(800, 550);
   }
   else
   {
      faxdoc= new ConvertIntoTextImage();
      ((ConvertIntoTextImage) faxdoc).text=faxeditor.getText();
      ((ConvertIntoTextImage) faxdoc).pageimage=createImage(800, 550);
      ((ConvertIntoTextImage) faxdoc).prepare();
   }
   /* Set modem configuration*/
   sendfax=new SendingFax();
   sendfax.bitorder=true;
   sendfax.log=true;
   sendfax.debug=d;
   sendfax.faxclass=1;
   sendfax.resolution=sendfax.RESOLUTION_NORMAL;
   sendfax.timeout=60;
   if (faxpropertypage.getmodemclass()=="Class 1") sendfax.faxclass=2;
   if (faxpropertypage.getDialing()=="Class 2") sendfax.faxclass=20;
   sendfax.dialingmode=faxpropertypage.getDialing();
   sendfax.flowcontrol=faxpropertypage.getFlowControlIn();
   if (faxpropertypage.getFlowcontrol()!="")
   {
      sendfax.flowcontrolin=(String)faxpropertypage.getFlowcontrol();
   }
   sendfax.setPortName(faxpropertypage.getCommPortName());
   sendfax.modemFBOR=true;
```

```
            if (modemstringtext.getText().length()>0) sendfax.setInitString(modemstringtext.getText());
            sendfax.listener=this;
            if (sendfax.openForFax(faxdoc))
            {
               if (sendfax.sendFax(faxtextfield.getText()))
               {
                  this.processlabel.setText("Fax successfully sent");
                  sendbutton.setEnabled(true);
                }
            }
         }
         /*
         This is executed when user clicks the Close button.
         */
         else if("cancel".equals(actioncommand))
         {
            if (sendfax!=null) sendfax.close();
            processlabel.setForeground(java.awt.Color.blue);
            processlabel.setText("Inactive...");
            sendbutton.setEnabled(true);
            closebutton.setEnabled(false);
            sendfax.serialPort.close();
         }
            /*
            This is executed when user clicks the Cancel button.
            */
            else if("close".equals(actioncommand))
            {
               System.exit(0);
            }
      }
      /*
      faxProgress - This method is called to show status of fax sending operation in the label.
      Parameters: status ,page
      Return Value:
       */
      public void faxProgress(int status,int page)
      {
         if (status==ST_CLOSE) processlabel.setText("Closing...");
         if (status==ST_CONNECTING) processlabel.setText("Connecting...");
         if (status==ST_CONVERT_FIlES) processlabel.setText("Converting fax files...");
         if (status==ST_INIT_MODEM) processlabel.setText("Initializing modem...");
         if (status==ST_OPEN_PORT) processlabel.setText("Opening port...");
         if (status==ST_SEND_PAGE) processlabel.setText("Sending page " + (int) (page +1));
         if (status==FaxStatusListener.ST_REC_PAGE) processlabel.setText("Receiving page " + (int) (pag
         if (status==FaxStatusListener.ST_REC_CALL) processlabel.setText("Call detected...");
         if (status==FaxStatusListener.ST_WAIT_CALL) processlabel.setText("Waiting for call...");
         this.paintAll(this.getGraphics());
      }
}
```

Download this listing.

In the above listing, the main() method creates an instance of the PortFax class. This class generates the main window of the Fax application, as shown in Figure 6-2:

Figure 6-2: The Main Window of the Fax Application

The File menu provides three menu items: Open, View Properties, and Exit. The Open command opens the file selected by the end user in the edit pane of the Fax application. The View Properties command displays a user interface to set the properties of the COM port. The Fax application uses the FaxPropertyPage.jave file to create the user interface. The Exit command terminates the Fax application.

The Modem Initial String text box allows an end user to set the initial command string for the modem. The Fax Number text box specifies the number to which the end user wants to send the fax. The HTML check box enables an end user to send the fax as an HTML page. If the HTML check box is not selected, the Fax application sends the Fax as a text image.

The various methods defined in the PortFax.java File are:

- createGUI():Creates the user interface of the Fax application.
- actionPerformed():Acts as an event listener and activates an appropriate class or method based on the button that the end user clicks. For example, when the end user selects the HTML check box and clicks the Send Fax button, the actionPerformed() method creates an object of the ConvertIntoHTML class, which converts the text file into an image of an HTML page. If the end user does not select the HTML check box and clicks the Send Fax button, the actionPerformed() method creates an object of the ConvertIntoTextImage class. This class converts the file that is open in the edit pane to a text image. When the end user clicks the Close button of the Fax application, the actionPerformed() method closes the main window.
- faxProgress():Displays the status of the Fax application.

◄ PREVIOUS NEXT ►

Converting the File into an Image

The ConvertIntoHTML.java and ConvertIntoTextImage.java files convert the file that is open in the user interface of the Fax application to images of HTML and text pages, respectively. The ConvertIntoHTML.java and ConvertIntoTextImage.java files implement the interface defined by the PreparedFaxDoc.java file.

Listing 6-2 shows the PreparedFaxDoc.java file:

Listing 6-2: The PreparedFaxDoc.java File

```
/*Imports required java.awt.Image class.*/
import java.awt.Image;
/*
interface PreparedFaxDoc - This interface declare an abstract method.
Methods:
   getFaxPage
*/
public interface PreparedFaxDoc
{
    public abstract Image getFaxPage(int i);
}
```

Download this listing.

In the above listing, the getFaxPage()method returns the converted image. The image returned by the getFaxPage()method shows the contents of the file sent as a fax.

The ConvertIntoTextImage.java file converts the file opened in the edit pane to a text image. This image contains the contents of the file.

Listing 6-3 shows the ConvertIntoTextImage.java file:

Listing 6-3: The ConvertIntoTextImage.java File

```
/*Imports required javax.comm package classes.*/
import java.awt.*;
/*
class ConvertIntoTextImage - This class is used to convert editpane Contents into text page
Constructor:
   ConvertIntoTextImage - This constructor initializes parameters
Methods:
   getFaxPage - Returns fax Page.
   getTextImage - Returns text Page.
getTextString - This method is called to convert String into String array.
prepare - This method is called to invoke getTextString method.
*/
public class ConvertIntoTextImage implements PreparedFaxDoc
{
public Font textfont;
       public String text;
       public int linespage;
       public int topmargin;
       public int leftmargin;
       public Image pageimage;
       private String textstring[];
       private int linenumber;
       public ConvertIntoTextImage()
       {
           textfont = new Font("Serif", 0, 12);
           text = "";
           linespage = 66;
           topmargin = 4;
           leftmargin = 4;
           pageimage = null;
           textstring = new String[1000];
           linenumber = 0;
    }
   /*
   getFaxPage - This method is called to return fax Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    public Image getFaxPage(int page)
    {
        if(page * linespage >= linenumber)
            return null;
        else
            return getTextImage(page);
    }
   /*
   getTextImage - This method is called to return text Page.
   Parameters: page - Variable of integer type.
```

```
   Return Value: Image
   */
    private Image getTextImage(int page)
    {
         Graphics g = pageimage.getGraphics();
         g.setColor(Color.white);
         g.fillRect(0, 0, pageimage.getWidth(null), pageimage.getHeight(null));
         g.setColor(Color.black);
         g.setFont(textfont);
         int lineH = g.getFontMetrics().getHeight();
         int charW = g.getFontMetrics().stringWidth("X");
         int linesImage = pageimage.getHeight(null) / lineH - topmargin;
         if(linesImage < linespage)
         linespage = linesImage;
         int currentY = lineH * topmargin;
         int currentLine = 0;
         for(int currentTotalLine = page * linespage; currentLine < linespage &&
         currentTotalLine < linenumber; currentTotalLine++)
         {
            if(textstring[currentTotalLine] != null)
            g.drawString(textstring[currentTotalLine], leftmargin * charW, currentY);
            currentY += lineH;
            currentLine++;
         }
         return pageimage;
       }
      /*
      prepare - This method is called to invoke getTextString method.
      Parameters:  NA
      Return Value: NA
      */
      public void prepare()
       {
        getTextString(text);
       }
      /*
      getTextString - This method is called to convert String into String array.
      Parameters: text -Object of String object.
      Return Value: NA
      */
       private void getTextString(String text)
       {
          int p = 0;
          linenumber = 0;
          for(p = text.indexOf("\n"); p >= 0; p = text.indexOf("\n"))
          {
            textstring[linenumber++] = text.substring(0, p);
            text = text.substring(p + 1, text.length());
          }
        textstring[linenumber++] = text;
    }
}
```

Download this listing.

In the above listing, the ConvertIntoTextImage class converts the text of the file into a image. The methods defined in the above listing are:

- `getFaxPage()`:Returns the page as an image to send as a fax. This method is an abstract method that is defined in the PreparedFaxDoc.java interface.
- `getTextImage()`:Creates an object of the Graphics class. The method converts the file opened in the edit pane to a text image.
- `prepare()`:Invokes the getTextString() method.
- `getTextString()`:Retrieves the contents of the text file that is open in the edit pane into a String array.

The ConvertIntoHTML.java file converts the file opened in the edit pane to an image of the HTML page sent as a fax.

Listing 6-4 shows the ConvertIntoHTML.java file:

Listing 6-4: The ConvertIntoHTML.java File

```
/*Imports required java.awt package classes*/
import java.awt.*;
/*Imports required javax.swing classes*/
import javax.swing.JComponent;
import javax.swing.JEditorPane;
import javax.swing.text.JTextComponent;
/*
class ConvertIntoHTML - This class is the use to convert editpane Contents into HTML image.
Constructor:
ConvertIntoHTML - This constructor initializes parameters
Methods:
   getFaxPage - Returns fax Page
   getHTMLPage - Returns HTML Page
```

```
*/
public class ConvertIntoHTML implements PreparedFaxDoc
{
    public String text[];
    public Image pageimage;
    private JEditorPane htmleditorpanel;
    public ConvertIntoHTML()
    {
        pageimage = null;
        htmleditorpanel = new JEditorPane();
    }
   /*
   getFaxPage - This method is called to return fax Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    public Image getFaxPage(int page)
    {
       if(text == null)
            return null;
        if(page >= text.length)
            return null;
        else
            return getHTMLPage(page);
    }
   /*
   getHTMLPage - This method is called to return HTML Page.
   Parameters: page - Variable of integer type.
   Return Value: Image
   */
    private Image getHTMLPage(int page)
    {
         Graphics g = pageimage.getGraphics();
         htmleditorpanel.setSize(pageimage.getWidth(null), pageimage.getHeight(null));
         htmleditorpanel.setBackground(Color.white);
         htmleditorpanel.setEditable(false);
         htmleditorpanel.setContentType("text/html");
         htmleditorpanel.setText(text[page]);
         htmleditorpanel.paint(g);
         return pageimage;
    }
}
```

Download this listing.

In the above listing, the ConvertIntoHTML.java file implements the PreparedFaxDoc.java interface. The various methods defined in the ConvertIntoHTML.java file are:

- `getFaxPage()`:Returns the page as an HTML page to send as a fax. This method is an abstract method that is defined in the PreparedFaxDoc.java interface.
- `getHTMLPage()`:Converts the HTML page of the text file open in the user interface of the Fax application to an image.

PREVIOUS NEXT

Displaying Port Properties

The FaxPropertyPage.java file displays the properties of the port to which the modem is connected. The modem helps send the fax to the fax number of the destination. The FaxPropertyPage.java file creates a user interface to display and set the properties of the port.

Listing 6-5 shows the FaxPropertyPage.java file:

Listing 6-5: The FaxPropertyPage.java File

```
/*Imports required Comm classes*/
import javax.comm.*;
/*Imports required javax.swing package classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required java.awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required java.util package classes*/
import java.util.*;
/*
class FaxPropertyPage - This class is the Port Property dialog window,
Enable the end user to change the property of port.
Constructor:
   FaxPropertyPage - This constructor creates GUI.
Methods:
   setParity - Sets Parity bit.
   setStopbits - Sets Stop bit.
   setDatabits - Set Data bit
   setBaudRate - Sets Baud rate.
getCancelButton - Returns object of cancel button.
   getOkButton -  Returns object of ok button.
getFlowControlOut - Returns flow control out value.
getFlowControlIn - Returns flow control in value.
   getParity - Returns Parity bit value.
   getDatabits - Returns Data bit value.
   getStopbits - Returns Stop bit value.
   getBaudRate - Returns Baud rate value.
   getCommPortName - Returns  port name.
   getCommPort -  Returns object of CommPort class
*/
class FaxPropertyPage extends JFrame implements ActionListener
{
   /*Declare object of Enumeration class.*/
   Enumeration listport;
   /*Declare object of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /*Declare object of JPanel class.*/
   JPanel buttonpanel;
   JPanel propertylist;
   /*Declare object of JLabel class.*/
   JLabel portlistlabel;
   JLabel flowlabel;
   JLabel paritylabel;
   JLabel dialinglabel;
   JLabel modemclasslabel;
   JLabel flowcontrollabel;
   JLabel stoplabel;
   /*Declare object of JComboBox class.*/
   JComboBox comportcombo;
   JComboBox flowcombo;
   JComboBox modemclasscombo;
   JComboBox dialingcombo;
   JComboBox flowcontrolcombo;
   /*
   Declare object of CommPortIdentifier class.
   */
   CommPortIdentifier portId;
   public FaxPropertyPage()
   {
      /* Set title of main window. */
      super("Fax Property Page");
      /*
      Declare and initialize object of Container class.
      */
      Container contentpane=getContentPane();
      comportcombo=new JComboBox();
      listport=CommPortIdentifier.getPortIdentifiers();
      while (listport.hasMoreElements())
      {
         portId=(CommPortIdentifier)listport.nextElement();
         if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
```

```
            comportcombo.addItem(portId.getName());
        }
    }
    /*Initialize Object of JComboBox. */
    flowcombo=new JComboBox();
    flowcombo.addActionListener(this);
    /*
    Declare and initialize object of GridBagLayout class
    */
    GridBagLayout gridbaglayout=new GridBagLayout();
    /*
    Declare and initialize object of GridBagConstraints class
    */
    GridBagConstraints gridbagconstraint=new GridBagConstraints();
    /*
    Initialize object of JPanel class and set its layout as gridbaglayout.
    */
    propertylist=new JPanel(gridbaglayout);
    /*Set background color as white.*/
    propertylist.setBackground(Color.white);
    /*
    Initialize object of JLabel class and gives its name.
    */
    Portlistlabel=new JLabel("Port Name",JLabel.RIGHT);
    /*
    Initialize object of JLabel class and gives its name.
    */
    flowlabel=new JLabel("Flow Control In");
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.fill=GridBagConstraints.HORIZONTAL;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbagconstraint.insets=new Insets(5,5,5,5);
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=0;
    gridbagconstraint.weightx=1.0;
    gridbagconstraint.weighty=1.0;
    gridbaglayout.setConstraints(portlistlabel,gridbagconstraint);
    /* Add portlistlabel lable to panel. */
    propertylist.add(portlistlabel);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
    gridbagconstraint.gridy=0;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbagconstraint.insets=new Insets(5,5,5,5);
    gridbaglayout.setConstraints(comportcombo,gridbagconstraint);
    /*Add comportcombo combo box to panel.*/
    propertylist.add(comportcombo);
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=1;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(flowlabel,gridbagconstraint);
    /*Add flowlabel lable to panel.*/
    propertylist.add(flowlabel);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
    gridbagconstraint.gridy=1;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(flowcombo,gridbagconstraint);
    /*Add comportcombo combo box to panel.*/
    propertylist.add(flowcombo);
    /*
    Initialize object of JLabel class and gives its name.
    */
    modemclasslabel=new JLabel("Modem Class",JLabel.RIGHT);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=0;
    gridbagconstraint.gridy=2;
    gridbagconstraint.anchor=GridBagConstraints.CENTER;
    gridbaglayout.setConstraints(modemclasslabel,gridbagconstraint);
    /*Add modemclasslabel label to panel.*/
    propertylist.add(modemclasslabel);
    /*
    Initialize object of JComboBox class.
    */
    modemclasscombo=new JComboBox();
    modemclasscombo.addActionListener(this);
    /*
    Set constraints for gridbagconstraint.
    */
    gridbagconstraint.gridx=1;
```

```
gridbagconstraint.gridy=2;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(modemclasscombo,gridbagconstraint);
/*Add comportcombo combo box to panel.*/
propertylist.add(modemclasscombo);
/*
Initialize object of JLabel class and gives its name.
*/
dialinglabel=new JLabel("Dialing",JLabel.RIGHT);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(dialinglabel,gridbagconstraint);
/*Add dialinglabel label to panel.*/
propertylist.add(dialinglabel);
/*
Initialize object of JComboBox class.
*/
dialingcombo=new JComboBox();
dialingcombo.addActionListener(this);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=0;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(dialingcombo,gridbagconstraint);
/*Add dialingcombo combo to panel.*/
propertylist.add(dialingcombo);
/*
Initialize object of JLabel class and gives its name.
*/
flowcontrollabel=new JLabel("Flow Control Command",JLabel.RIGHT);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=2;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrollabel,gridbagconstraint);
/*
Add flowcontrollabel label to panel.
*/
propertylist.add(flowcontrollabel);
/*
Initialize object of JComboBox class.
*/
flowcontrolcombo=new JComboBox();
flowcontrolcombo.addActionListener(this);
/*
Set constraints for gridbagconstraint.
*/
gridbagconstraint.gridx=3;
gridbagconstraint.gridy=1;
gridbagconstraint.anchor=GridBagConstraints.CENTER;
gridbaglayout.setConstraints(flowcontrolcombo,gridbagconstraint);
/*
Add flowcontrolcombo combo to panel.
*/
propertylist.add(flowcontrolcombo);
contentpane.add(propertylist);
/*Fill dialingcombo combo box.*/
dialingcombo.addItem("Tone");
dialingcombo.addItem("Pulse");
dialingcombo.setSelectedItem("Tone");
/*Fill flowcombo combo box.*/
flowcombo.addItem("none");
flowcombo.addItem("RtsCts");
flowcombo.addItem("Xon/Xoff In");
flowcombo.setSelectedItem("none");
/*Fill flowcontrolcombo combo box.*/
flowcontrolcombo.addItem("");
flowcontrolcombo.addItem("-- RtsCts --");
flowcontrolcombo.addItem("-- XonXoff --");
flowcontrolcombo.addItem("AT+FLO=2");
flowcontrolcombo.addItem("AT&K3");
flowcontrolcombo.addItem("AT&\\Q3");
flowcontrolcombo.addItem("AT+FLO=1");
flowcontrolcombo.addItem("AT&K4");
flowcontrolcombo.addItem("AT&\\Q4");
flowcontrolcombo.setSelectedItem("");
/*Fill modemclasscombo combo box.*/
modemclasscombo.addItem("Class 1");
modemclasscombo.addItem("Class 2");
```

```
   modemclasscombo.addItem("Class 2.0");
   modemclasscombo.setSelectedItem("Class 1");
   /*
   Initialize object of JButton and set its label.
   */
   okbutton=new JButton("OK");
   okbutton.setActionCommand("ok");
   okbutton.addActionListener(new ActionListener()
   {public void actionPerformed(ActionEvent ae){setVisible(false);}});
   /*
   Initialize object of JButton and set its label
   */
   cancelbutton=new JButton("Cancel");
   okbutton.addActionListener(new ActionListener()
   {public void actionPerformed(ActionEvent ae){setVisible(false);}});
   cancelbutton.setActionCommand("cancel");
   cancelbutton.addActionListener(this);
   /* Initialize object of JPanel. */
   buttonpanel=new JPanel();
   /*
   Set buttonpanel  layout as gridbaglayout.
   */
   buttonpanel.setLayout(gridbaglayout);
   /*
   Set constraints for gridbagconstraint.
   */
   gridbagconstraint.fill = GridBagConstraints.HORIZONTAL;
   gridbagconstraint.insets=new Insets(5,5,5,5);
   gridbagconstraint.gridx=0;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(okbutton,gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(okbutton);
   /*
   Set constraints for gridbagconstraint.
   */
   gridbagconstraint.gridx=1;
   gridbagconstraint.gridy=0;
   gridbagconstraint.anchor=GridBagConstraints.CENTER;
   gridbaglayout.setConstraints(cancelbutton,gridbagconstraint);
   /*Add button to panel*/
   buttonpanel.add(cancelbutton);
   contentpane.add(buttonpanel,BorderLayout.SOUTH);
   addWindowListener(new WindowAdapter()
   {
      public void windowClosed(WindowEvent we)
      {
         setVisible(false);
      }
   });
   /*Set frame size.*/
   setSize(400,200);
}
/*
actionPerformed - This methods is executed when the end user selects items from combo box or clic
Parameters: e - Object of ActionEvent class
Return Value: NA
*/
public void actionPerformed(ActionEvent e)
{
   if (e.getSource() instanceof JButton)
   {
      setVisible(false);
   }
   else
   {
      JComboBox cb = (JComboBox)e.getSource();
      String petName = (String)cb.getSelectedItem();
   }
}
/*
getCommPort - Gives object of  CommPort class
Parameters: NA
Return Value: CommPortIdentifier
*/
public CommPortIdentifier getCommPort()
{
   CommPortIdentifier comp=null;
   try
   {
      comp= javax.comm.CommPortIdentifier.getPortIdentifier((String)comportcombo.getSelectedItem(
   }
   catch(NoSuchPortException e)
   {
      JOptionPane.showMessageDialog(null,"open","Port not exists",JOptionPane.PLAIN_MESSAGE);
   }
```

```
   return comp;
}
/*
getCommPortName - Gives name of selected port.
Parameters: NA
Return Value: String
*/
public String getCommPortName()
{
   return (String)comportcombo.getSelectedItem();
}
/*
getDialing - Gives dialing mode.
Parameters: NA
Return Value: String
*/
public String getDialing()
{
   return (String)dialingcombo.getSelectedItem();
}
/*
getmodemclass - Gives modem class.
Parameters: NA
Return Value: String
*/
public String getmodemclass()
{
   return (String)modemclasscombo.getSelectedItem();
}
/*
getFlowControlIn - Gives type of flow in control.
Parameters: NA
Return Value: int
*/
public int getFlowControlIn()
{
   String flowcontrol=(String)flowcombo.getSelectedItem();
   if (flowcontrol.equals("none"))
   {
      return 0;
   }
   if (flowcontrol.equals("RtsCts"))
   {
      return 2;
   }
   if (flowcontrol.equals("Xon/Xoff In"))
   {
      return 1;
   }
   return 0;
}
/*
getCancelButton - Gives object of JButton class.
Parameters: NA
Return Value: JButton
*/
public JButton getCancelButton()
{
   return cancelbutton;
}

public void setCommPort(String commname)
{
   comportcombo.setSelectedItem(commname);
}
public void setdialingMode(String dialing)
{
   dialingcombo.setSelectedItem(dialing);
}
public void setModemClass(String modemclass)
{
   modemclasscombo.setSelectedItem(modemclass);
}
public void setFlowControl(String flowcontrol)
{
   flowcontrolcombo.setSelectedItem(flowcontrol);
}
public void setFlowControlIn(int flowcontrolin)
{
   if(flowcontrolin==0)
   {
      flowcombo.setSelectedItem("none");
   }
   else if (flowcontrolin==1)
   {
      flowcombo.setSelectedItem("RtsCts")
   }
```

```
            else
            {
               flowcombo.setSelectedItem("Xon/Xoff In");
            }
        }
        /*
        getFlowcontrol - Gives flow control.
        Parameters: NA
        Return Value: String
        */
        public String getFlowcontrol()
        {
            return (String)flowcontrolcombo.getSelectedItem();
        }
}
```

Download this listing.

In the above listing, the constructor of the FaxPropertyPage class creates the user interface of the FaxPropertyPage.java file, as shown in Figure 6-3:

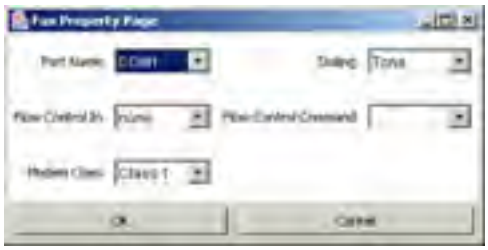

Figure 6-3: The Fax Property Page Window

The various methods defined in the FaxPropertyPage.java file are:

- `actionPerformed()`:Acts as an event listener and activates an appropriate class or method based on the button that an end user clicks. When the end user clicks the OK button, the actionPerformed() method sets the properties of the port to the values selected on the Fax Property Page window that the FaxPropertyPage class creates. When an end user clicks the Cancel button, the actionPerformed() method closes the Fax Property Page window.
- `getCommPort()`: Determines whether the selected COM port exists or not.
- `setCommPort()`:Sets the value of the selected COM port.
- `getDialing()`:Returns the dialing mode of the selected COM port, such as?.
- `setdialingMode()`:Sets the dialing mode property for the selected COM port.
- `getmodemclass()`:Returns the class of the modem that helps send the fax.
- `setModemClass()`:Sets the value of the modem class to the selected value.
- `getFlowcontrol()`:Returns the value of the Flow control property of the selected COM port. The Flow control property indicates data flow from the modem to the selected COM port.
- `setFlowControl()`:Sets the value of the Flow Control property of the selected COM port.
- `getFlowControlIn()`:Returns the value of the Flow Control In property of the selected COM port. The Flow Control In property indicates data flow from the selected COM port to the modem.
- `setFlowControlIn`:Sets the value of the Flow Control In property of the selected COM port.
- `getCommPortName()` Returns the name of the COM port that sends the fax.
- `getCancelButton()`:Creates and returns an instance of the Cancel button.

Displaying the Status of the Fax Application

The FaxStatusListener.java file sets integer values for the status of the Fax application.

Listing 6-6 shows the FaxStatusListener.java file:

Listing 6-6: The FaxStatusListener.java File

```
public interface FaxStatusListener
{
    public abstract void faxProgress(int i, int j);
    public static final int ST_OPEN_PORT = 1;
    public static final int ST_INIT_MODEM = 2;
    public static final int ST_CONNECTING = 3;
    public static final int ST_SEND_PAGE = 4;
    public static final int ST_CLOSE = 5;
    public static final int ST_CONVERT_FIlES = 6;
    public static final int ST_REC_PAGE = 4;
    public static final int ST_REC_CALL = 7;
    public static final int ST_WAIT_CALL = 8;
}
```

Download this listing.

In the above listing, the FaxStatusListener.java file declares an abstract method, faxProgress(). This method accepts two integer parameters. The first parameter indicates the status of the fax and the second parameter indicates the page to send as a fax. The PortFax.java file implements the FaxStatusListener interface to display the current status of the Fax application.

Setting Modem Capabilities

The ModemCapabilities.java file sets the encoding and decoding capabilities of a modem.

Listing 6-7 shows the ModemCapabilities.java file:

Listing 6-7: The ModemCapabilities.java File

```
/*Imports required java.util.Vector class*/
import java.util.Vector;
/*
class ModemCapabilities - This class is the main class of the application.
This class initializes the interface and loads all components like the menubar,
textareas, and buttons before displaying the result.
Constructor:
ModemCapabilities - This constructor initializes various modem parameters
Methods:
encodeCapabilities - This method is called to ecode modem capabilities
decodeCapabilitiesClass2 - Find modem capabilities of class2 type.
decodeCapabilities -  Find modem capabilities of class1 type.
*/
public class ModemCapabilities
{
public String vRate;
public String recRate;
   public boolean t4;
       public int resolution;
       public int rate;
       public int length;
       public int width;
       public int huffman;
       public int scanTime;
       public int errorCorrection;
       public boolean binaryTransfer;
       private static int itemlength = 0;
       private static int itemsize = 1;
       private static int seconditemsize = 2;
       private static int thirditemsize = 3;
       private static int forthitemsize = 4;
       private static int fifthitemsize = 5;
       private static int sixthitemsize = 6;
       private static int seventhitemsize = 7;
       public ModemCapabilities()
       {
           t4 = true;
           resolution = 0;
           rate = 96;
           vRate = "V29";
           recRate = "";
           length = 0;
           width = 1728;
           huffman = 1;
           scanTime = 0;
           errorCorrection = 0;
           binaryTransfer = false;
       }
      /*
      encodeCapabilities - This method is called to ecode modem capabilities
      Parameters: NA
      Return Value: byte[]
      */
       public byte[] encodeCapabilities()
       {
            byte bi[] = new byte[3];
          bi[0] = 0;
          bi[1] = 0;
          bi[2] = 0;
           bi[1] = 64;
                byte tmp = 0;
           if(rate == 145 && vRate.compareTo("V17") == 0)
                tmp = 4;
           if(rate == 145 && vRate.compareTo("V33") == 0)
                tmp = 8;
           if(rate == 121 && vRate.compareTo("V17") == 0)
                tmp = 20;
           if(rate == 121 && vRate.compareTo("V33") == 0)
                tmp = 24;
           if(rate == 96 && vRate.compareTo("V17") == 0)
                tmp = 36;
           if(rate == 72 && vRate.compareTo("V17") == 0)
                 tmp = 52;
           if(rate == 72 && vRate.compareTo("V33") == 0)
                 tmp = 48;
           if(rate == 72 && vRate.compareTo("V29") == 0)
```

```
                tmp = 48;
          if(rate == 96 && vRate.compareTo("V29") == 0)
          tmp = 32;
        if(rate == 48)
          tmp = 16;
        if(rate == 24)
          tmp = 0;
        tmp &= 0xff;
        bi[1] = (byte)(bi[1] | tmp);
        if(resolution == 1)
           bi[1] = (byte)(bi[1] | 2);
        if(huffman == 2)
             bi[1] = (byte)(bi[1] | 1);
         tmp = 0;
         if(width == 1728)
             tmp = 0;
         if(width == 2432)
             tmp = 64;
         if(width == 2048)
             tmp = -128;
         tmp &= 0xff;
         bi[2] = (byte)(bi[2] | tmp);
         tmp = 0;
         if(length == 2)
             tmp = 16;
         if(length == 1)
             tmp = 32;
         tmp &= 0xff;
        bi[2] = (byte)(bi[2] | tmp);
         tmp = 0;
         if(scanTime == 20)
             tmp = 0;
         if(scanTime == 40)
             tmp = 2;
         if(scanTime == 10)
             tmp = 4;
         if(scanTime == 5)
             tmp = 8;
         if(scanTime == 0)
             tmp = 14;
         tmp &= 0xff;
         bi[2] = (byte)(bi[2] | tmp);
         return bi;
 }
/*
decodeCapabilitiesClass2 - This method is called to decode capabilities of class2 type mode
Parameters: ae - an ActionEvent object containing details of the event.
Return Value:boolean
 */
public boolean decodeCapabilitiesClass2(String str)
{
   Vector items = new Vector();
   String item = "";
   for(int p = str.indexOf(","); p >= 0; p = str.indexOf(","))
   {
         item = "";
         if(p > 0)
             item = str.substring(0, p);
         str = str.substring(p + 1, str.length());
         items.addElement(item);
     }
     if(items.size() > itemlength)
     {
         item = (String)items.elementAt(itemlength);
          if(item.trim().equals("0"))
          resolution = 0;
          if(item.trim().equals("1"))
          resolution = 1;
    }
     if(items.size() > itemsize)
    {
         item = (String)items.elementAt(itemsize);
         if(item.trim().equals("0"))
         {
             rate = 24;
             vRate = "V27";
         }
         if(item.trim().equals("1"))
         {
             rate = 48;
             vRate = "V27";
         }
         if(item.trim().equals("2"))
         {
             rate = 72;
             vRate = "V29";
         }
         if(item.trim().equals("3"))
```

```
                    {
                        rate = 96;
                        vRate = "V29";
                    }
                    if(item.trim().equals("4"))
                    {
                        rate = 121;
                        vRate = "V17";
                    }
                    if(item.trim().equals("5"))
                    {
                        rate = 145;
                        vRate = "V17";
                    }
                }
                if(items.size() > seconditemsize)
                {
                    item = (String)items.elementAt(seconditemsize);
                    width = 1728;
                    if(item.trim().equals("0"))
                    width = 1728;
                    if(item.trim().equals("1"))
                    width = 2048;
                    if(item.trim().equals("2"))
                    width = 2432;
            }
            if(items.size() > thirditemsize)
            {
                item = (String)items.elementAt(thirditemsize);
                if(item.trim().equals("0"))
                length = 0;
                if(item.trim().equals("1"))
                length = 1;
                if(item.trim().equals("2"))
                length = 2;
            }
            if(items.size() > forthitemsize)
            {
                item = (String)items.elementAt(forthitemsize);
                huffman = 2;
                if(item.trim().equals("0"))
                huffman = 1;
            }
            if(items.size() > fifthitemsize)
            {
                item = (String)items.elementAt(fifthitemsize);
                errorCorrection = 0;
                if(item.trim().equals("1"))
                errorCorrection = 1;
                if(item.trim().equals("2"))
                errorCorrection = 2;
            }
            if(items.size() > sixthitemsize)
            {
                item = (String)items.elementAt(sixthitemsize);
                binaryTransfer = false;
                if(item.trim().equals("1"))
                binaryTransfer = true;
            }
            if(items.size() > seventhitemsize)
            {
                item = (String)items.elementAt(seventhitemsize);
                if(resolution == 0)
                {
                    if(item.trim().equals("0"))
                        scanTime = 0;
                    if(item.trim().equals("1"))
                        scanTime = 5;
                    if(item.trim().equals("2"))
                        scanTime = 10;
                    if(item.trim().equals("3"))
                        scanTime = 20;
                    if(item.trim().equals("4"))
                        scanTime = 20;
                    if(item.trim().equals("5"))
                        scanTime = 20;
                    if(item.trim().equals("6"))
                        scanTime = 40;
                    if(item.trim().equals("7"))
                        scanTime = 40;
                }
else
                {
                    if(item.trim().equals("0"))
                        scanTime = 0;
                    if(item.trim().equals("1"))
                        scanTime = 5;
                    if(item.trim().equals("2"))
```

```
                scanTime = 10;
            if(item.trim().equals("3"))
                scanTime = 10;
            if(item.trim().equals("4"))
                scanTime = 10;
            if(item.trim().equals("5"))
                scanTime = 20;
            if(item.trim().equals("6"))
                scanTime = 20;
            if(item.trim().equals("7"))
                scanTime = 40;
        }
    }
    return true;
 }
/*
decodeCapabilities - This method is called to decode capabilities of class1 type mode
Parameters:   ae - an ActionEvent object containing details of the event.
Return Value:boolean
 */
 public boolean decodeCapabilities(byte bi[])
 {
    int b = bi[1];
    b &= 0x40;
    if(b == 64)
        t4 = true;
    else
        t4 = false;
    b = bi[1];
    b &= 0x3c;
    rate = 96;
    vRate = "V29";
    if(b == 56)
    {
        rate = 96;
        vRate = "V33";
    }
    if(b == 52)
    {
        rate = 145;
        vRate = "V17";
    }
    if(b == 48)
    {
        rate = 72;
        vRate = "V29";
    }
    if(b == 32)
    {
        rate = 96;
        vRate = "V29";
    }
    if(b == 16)
    {
        rate = 48;
        vRate = "V17";
    }
    if(b == 0)
    {
        rate = 24;
        vRate = "V27FB";
    }
    b = bi[1];
    b &= 2;
    if(b == 2)
        resolution = 1;
    b = bi[1];
    b &= 1;
    if(b == 1)
        huffman = 2;
    b = bi[2];
    b &= 0xc0;
    if(b == 0)
        width = 1728;
    if(b == 64)
        width = 2432;
    if(b == 128)
        width = 2048;
    if(b == 192)
        width = 2432;
    b = bi[2];
    b &= 0x30;
    if(b == 0)
        length = 0;
    if(b == 16)
        length = 2;
    if(b == 32)
        length = 1;
```

```
            b = bi[2];
            b &= 0xe;
            if(b == 0)
                scanTime = 20;
            if(b == 2)
                scanTime = 40;
            if(b == 4)
                scanTime = 10;
            if(b == 6)
                scanTime = 10;
            if(b == 8)
                scanTime = 5;
            if(b == 14)
                scanTime = 0;
            if(b == 10)
                scanTime = 40;
            if(b == 12)
                scanTime = 20;
            return true;
        }
}
```

Download this listing.

In the above listing, the ModemCapabilities class sets the encoding and decoding capabilities of a modem, such as time to scan the document and transfer rate to send the fax. The various methods defined in the Capabilities class are:

- `encodeCapabilities()`:Sets the encoding capabilities of a modem.
- `decodeCapabilitiesClass2()`:Sets the decoding capabilities of a class 2 modem.
- `decodeCapabilities()`:Sets the decoding capabilities of a modem.

PREVIOUS NEXT

Encoding the Fax Document

The ImageToFaxEncoder.java encodes the fax document that is currently opened in the edit pane of the Fax Application window.

Listing 6-8 shows the ImageToFaxEncoder.java file:

Listing 6-8: The ImageToFaxEncoder.java File

```
import java.awt.Image;
import java.awt.image.PixelGrabber;
import java.io.PrintStream;
public class ImageToFaxEncoder
{
public boolean debug;
       public int minBytesLine;
       public boolean alignEOL;
       public boolean completeLine;
       public int lineWidth;
       public int pageLength;
       public boolean completePage;
       public boolean scaleImage;
       public int scaleFactor;
       public boolean centerImage;
       public int whiteColor;
       public int blackColor;
       private int EOLimagearray[] = {
         86, 101, 86, 102, 86, 103, 86, 104, 86, 105,86, 106, 86, 107, 86, 108, 86, 114, 86, 115,
         86, 116, 86, 117, 86, 118, 86, 119, 86, 120,86, 121, 86, 122, 86, 123, 86, 124, 86, 130,
         86, 131, 86, 132, 86, 138, 86, 139, 86, 150,86, 151, 87, 101, 87, 102, 87, 103, 87, 104,
         87, 105, 87, 106, 87, 107, 87, 108, 87, 109,87, 110, 87, 114, 87, 115, 87, 116, 87, 117,
         87, 118, 87, 119, 87, 120, 87, 121, 87, 122,87, 123, 87, 124, 87, 130, 87, 131, 87, 132,
         87, 138, 87, 139, 87, 140, 87, 149, 87, 150,87, 151, 88, 101, 88, 102, 88, 109, 88, 110,
         88, 114, 88, 115, 88, 129, 88, 130, 88, 132,88, 133, 88, 139, 88, 140, 88, 141, 88, 148,
         88, 149, 88, 150, 89, 101, 89, 102, 89, 110,89, 111, 89, 114, 89, 115, 89, 129, 89, 130,
         89, 132, 89, 133, 89, 140, 89, 141, 89, 142,89, 147, 89, 148, 89, 149, 90, 101, 90, 102,
         90, 110, 90, 111, 90, 114, 90, 115, 90, 128,90, 129, 90, 133, 90, 134, 90, 141, 90, 142,
         90, 143, 90, 146, 90, 147, 90, 148, 91, 101,91, 102, 91, 110, 91, 111, 91, 114, 91, 115,
         91, 128, 91, 129, 91, 133, 91, 134, 91, 142,91, 143, 91, 144, 91, 145, 91, 146, 91, 147,
         92, 101, 92, 102, 92, 109, 92, 110, 92, 114,92, 115, 92, 116, 92, 117, 92, 118, 92, 119,
         92, 120, 92, 121, 92, 122, 92, 123, 92, 128,92, 129, 92, 133, 92, 134, 92, 143, 92, 144,
         92, 145, 92, 146, 93, 101, 93, 102, 93, 103,93, 104, 93, 105, 93, 106, 93, 107, 93, 108,
         93, 109, 93, 110, 93, 114, 93, 115, 93, 116,93, 117, 93, 118, 93, 119, 93, 120, 93, 121,
         93, 122, 93, 123, 93, 127, 93, 128, 93, 134,93, 135, 93, 143, 93, 144, 93, 145, 93, 146,
         94, 101, 94, 102, 94, 103, 94, 104, 94, 105,94, 106, 94, 107, 94, 108, 94, 109, 94, 110,
         94, 114, 94, 115, 94, 127, 94, 128, 94, 129,94, 130, 94, 131, 94, 132, 94, 133, 94, 134,
         94, 135, 94, 142, 94, 143, 94, 144, 94, 145,94, 146, 94, 147, 95, 101, 95, 102, 95, 110,
         95, 111, 95, 114, 95, 115, 95, 126, 95, 127,95, 128, 95, 129, 95, 130, 95, 131, 95, 132,
         95, 133, 95, 134, 95, 135, 95, 136, 95, 141,95, 142, 95, 143, 95, 146, 95, 147, 95, 148,
         96, 101, 96, 102, 96, 110, 96, 111, 96, 114,96, 115, 96, 126, 96, 127, 96, 135, 96, 136,
         96, 140, 96, 141, 96, 142, 96, 147, 96, 148,96, 149, 97, 101, 97, 102, 97, 110, 97, 111,
         97, 114, 97, 115, 97, 126, 97, 127, 97, 135,97, 136, 97, 139, 97, 140, 97, 141, 97, 148,
         97, 149, 97, 150, 98, 101, 98, 102, 98, 110,98, 111, 98, 114, 98, 115, 98, 125, 98, 126,
         98, 136, 98, 137, 98, 138, 98, 139, 98, 140,98, 149, 98, 150, 98, 151, 99, 101, 99, 102,
         99, 111, 99, 112, 99, 114, 99, 115, 99, 125,99, 126, 99, 136, 99, 137, 99, 138, 99, 139,
         99, 150, 99, 151
    };
     public boolean createEOP;
    private int filterarrar[][] =
    {
       {
       15, 10, 64
       },
    {
       200, 12, 128
    },
    {
        201, 12, 192
    },
    {
        91, 12, 256
    },
    {
        51, 12, 320
    },
    {
        52, 12, 384
    },
    {
        53, 12, 448
    },
    {
        108, 13, 512
    },
    {
```

```
    109, 13, 576
},
{
   74, 13, 640
},
{
   75, 13, 704
},
{
   76, 13, 768
},
{
   77, 13, 832
},
{
   114, 13, 896
},
{
   115, 13, 960
},
{
   116, 13, 1024
},
{
   117, 13, 1088
},
{
   118, 13, 1152
},
{
   119, 13, 1216
},
{
   82, 13, 1280
},
{
   83, 13, 1344
},
{
   84, 13, 1408
},
{
   85, 13, 1472
},
{
   90, 13, 1536
},
{
   91, 13, 1600
},
{
   100, 13, 1664
},
{
   101, 13, 1728
}
};
private int matharray[][] = {
{
   27, 5, 64
},
{
   18, 5, 128
},
{
   23, 6, 192
},
{
   55, 7, 256
},
{
   54, 8, 320
},
{
   55, 8, 384
},
{
   100, 8, 448
},
{
   101, 8, 512
},
{
   104, 8, 576
},
{
   103, 8, 640
```

```
},
{
   204, 9, 704
},
{
   205, 9, 768
},
{
   210, 9, 832
},
{
   211, 9, 896
},
{
   212, 9, 960
},
{
   213, 9, 1024
},
{
   214, 9, 1088
},
{
   215, 9, 1152
},
{
   216, 9, 1216
},
{
   217, 9, 1280
},
{
   218, 9, 1344
},
{
   219, 9, 1408
},
{
   152, 9, 1472
},
{
   153, 9, 1536
},
{
   154, 9, 1600
},
{
   24, 6, 1664
},
{
   155, 9, 1728
}
};
private int whitcodearray[][] = {
{
   55, 10, 0
},
{
   2, 3, 1
},
{
   3, 2, 2
},
{
   2, 2, 3
},
{
   3, 3, 4
},
{
   3, 4, 5
},
{
   2, 4, 6
},
{
   3, 5, 7
},
{
   5, 6, 8
},
{
   4, 6, 9
},
{
   4, 7, 10
},
```

```
{
   5, 7, 11
},
{
   7, 7, 12
},
{
   4, 8, 13
},
{
   7, 8, 14
},
{
   24, 9, 15
},
{
   23, 10, 16
},
{
   24, 10, 17
},
{
   8, 10, 18
},
{
   103, 11, 19
},
{
   104, 11, 20
},
{
   108, 11, 21
},
{
   55, 11, 22
},
{
   40, 11, 23
},
{
   23, 11, 24
},
{
   24, 11, 25
},
{
   202, 12, 26
},
{
   203, 12, 27
},
{
   204, 12, 28
},
{
   205, 12, 29
},
{
   104, 12, 30
},
{
   105, 12, 31
},
{
   106, 12, 32
},
{
   107, 12, 33
},
{
   210, 12, 34
},
{
   211, 12, 35
},
{
   212, 12, 36
},
{
   213, 12, 37
},
{
   214, 12, 38
},
{
   215, 12, 39
},
{
```

```
        108, 12, 40
    },
    {
        109, 12, 41
    },
    {
        218, 12, 42
    },
    {
        219, 12, 43
    },
    {
        84, 12, 44
    },
    {
        85, 12, 45
    },
    {
        86, 12, 46
    },
    {
        87, 12, 47
    },
    {
        100, 12, 48
    },
    {
        101, 12, 49
    },
    {
        82, 12, 50
    },
    {
        83, 12, 51
    },
    {
        36, 12, 52
    },
    {
        55, 12, 53
    },
    {
        56, 12, 54
    },
    {
        39, 12, 55
    },
    {
        40, 12, 56
    },
    {
        88, 12, 57
    },
    {
        89, 12, 58
    },
    {
        43, 12, 59
    },
    {
        44, 12, 60
    },
    {
        90, 12, 61
    },
    {
        102, 12, 62
    },
    {
        103, 12, 63
    }
    };
    private int codearray[][] = {
    {
        53, 8, 0
    },
    {
        7, 6, 1
    },
    {
        7, 4, 2
    },
    {
        8, 4, 3
    },
    {
        11, 4, 4
    },
```

```
{
   12, 4, 5
},
{
   14, 4, 6
},
{
   15, 4, 7
},
{
   19, 5, 8
},
{
   20, 5, 9
},
{
   7, 5, 10
},
{
   8, 5, 11
},
{
   8, 6, 12
},
{
   3, 6, 13
},
{
   52, 6, 14
},
{
   53, 6, 15
},
{
   42, 6, 16
},
{
   43, 6, 17
},
{
   39, 7, 18
},
{
   12, 7, 19
},
{
   8, 7, 20
},
{
   23, 7, 21
},
{
   3, 7, 22
},
{
   4, 7, 23
},
{
   40, 7, 24
},
{
   43, 7, 25
},
{
   19, 7, 26
},
{
   36, 7, 27
},
{
   24, 7, 28
},
{
   2, 8, 29
},
{
   3, 8, 30
},
{
   26, 8, 31
},
{
   27, 8, 32
},
{
   18, 8, 33
},
{
```

```
      19, 8, 34
   },
   {
      20, 8, 35
   },
   {
      21, 8, 36
   },
   {
      22, 8, 37
   },
   {
      23, 8, 38
   },
   {
      40, 8, 39
   },
   {
      41, 8, 40
   },
   {
      42, 8, 41
   },
   {
      43, 8, 42
   },
   {
      44, 8, 43
   },
   {
      45, 8, 44
   },
   {
      4, 8, 45
   },
   {
      5, 8, 46
   },
   {
      10, 8, 47
   },
   {
      11, 8, 48
   },
   {
      82, 8, 49
   },
   {
      83, 8, 50
   },
   {
      84, 8, 51
   },
   {
      85, 8, 52
   },
   {
      36, 8, 53
   },
   {
      37, 8, 54
   },
   {
      88, 8, 55
   },
   {
      89, 8, 56
   },
   {
      90, 8, 57
   },
   {
      91, 8, 58
   },
   {
      74, 8, 59
   },
   {
      75, 8, 60
   },
   {
      50, 8, 61
   },
   {
      51, 8, 62
   },
   {
      52, 8, 63
```

```
}
};
 private byte imagebit[];
 private int leftshift;
 private byte masking;
 private int imagearrcount;
 private byte bytearray[];
 public ImageToFaxEncoder()
 {
     debug = false;
     minBytesLine = 32;
     alignEOL = true;
     completeLine = true;
     lineWidth = 1728;
     pageLength = 2387;
     completePage = false;
     scaleImage = false;
     scaleFactor = 1;
     centerImage = false;
     whiteColor = -1;
     blackColor = 0;
     createEOP = false;
     imagebit = new byte[0x30d40];
     leftshift = 7;
     masking = 0;
     imagearrcount = 0;
     bytearray = new byte[0];
 }
 private void setBitMap(int b)
 {
     byte bitMask = 0;
     bitMask = (byte)(1 << leftshift);
     if(b == 1)
         masking = (byte)(masking | bitMask);
        if(debug)
        System.out.print("".concat(String.valueOf(String.valueOf(b))));
        leftshift--;
        if(leftshift < 0)
        {
            leftshift = 7;
            imagebit[imagearrcount] = masking;
            imagearrcount++;
            masking = 0;
        }
    }
    private void setBitMask(int code, int bits)
    {
      int mask = 0;
      for(mask = 1 << bits - 1; mask != 0; mask >>= 1)
      if((code & mask) > 0)
         setBitMap(1);
      else
         setBitMap(0);
      if(debug)
               System.out.print(" ");
    }
   private void checkBitMap()
   {
      if(alignEOL)
      while(leftshift != 3)
      setBitMap(0);
      for(int h = 0; h < 11; h++)
      setBitMap(0);
      setBitMap(1);
    }
    private void _$41684()
    {
         for(int h = 0; h < 6; h++)
         checkBitMap();
    }
   public void addFiller(int runl, boolean inWhite)
   {
    int codes[][];
    int code[];
    if(runl >= 64)
    {
         if(inWhite)
         codes = matharray;
         else
         codes = filterarrar;
         code = codes[(int)Math.floor(runl / 64 - 1)];
         runl -= code[2];
         setBitMask(code[0], code[1]);
     }
     if(inWhite)
         codes = codearray;
     else
         codes = whitcodearray;
```

```
        code = codes[runl];
        setBitMask(code[0], code[1]);
    }
    public byte[] encodeImage(Image i)
    {
     int scale = scaleFactor;
     int currentPageRow = 0;
     if(scaleImage)
     {
          int w = i.getWidth(null);
          int h = i.getHeight(null);
          for(int j = 2; j < 10 && j * w <= lineWidth && j * h <= pageLength; j++)
          scale = j;
       }
        imagearrcount = 0;
        leftshift = 7;
        masking = 0;
        checkBitMap();
        int iw = i.getWidth(null);
        int ih = i.getHeight(null);
        int pix[] = new int[iw * ih];
        String s = "";
        String c = "";
        int runl = 0;
        int code[] = null;
        int lineBytesOffset = 0;
        PixelGrabber pg = new PixelGrabber(i, 0, 0, iw, ih, pix, 0, iw);
        try
        {
            pg.grabPixels();
        }
        catch(Exception e)
        {
            System.err.println("".concat(String.valueOf(String.valueOf(e.getMessage()))));
        }
         int bp = 1;
         if(bp == whiteColor)
         bp = 0;
         for(int h = 0; h < EOLimagearray.length; h += 2)
         pix[EOLimagearray[h] * iw + EOLimagearray[h + 1]] = bp;
         for(int row = 0; row < ih; row++)
         {
               for(int sc = 1; sc <= scale; sc++)
               {
                     currentPageRow++;
                     int ocol = 0;
                     int totalRow = 0;
                     lineBytesOffset = imagearrcount;
                     for(int col = 0; col < iw;)
                     {
                          runl = 0;
                        if(col == 0 && completeLine && centerImage)
                        {
                           totalRow = (lineWidth - iw * scale) / 2;
                           runl = totalRow;
                        }
                        while(col < lineWidth && col < iw && pix[row * iw + col] == whiteColor)
                        {
                            runl += scale;
                            col++;
                            totalRow += scale;
                        }
                        if(col >= iw && completeLine)
                        {
                           runl += lineWidth - totalRow;
                           totalRow = lineWidth;
                       }
                       if(runl >= 64)
                       {
                           code = matharray[(int)Math.floor(runl / 64 - 1)];
                           runl -= code[2];
                           if(debug)
                           System.out.println(String.valueOf(String.valueOf(
                           (new StringBuffer("White length=")).append(code[2]).append
                           (" code=").append(code[0]).append(" bits=").append(code[1]))));
                           setBitMask(code[0], code[1]);
                       }
                       code = codearray[runl];
                       if(debug)
                        System.out.println(String.valueOf(String.valueOf(
                        (new StringBuffer("White length=")).append(runl).append
                        (" code=").append(code[0]).append(" bits=").append(code[1]))));
                        setBitMask(code[0], code[1]);
                        if(col >= lineWidth || col >= iw)
                        break;
                        runl = 0;
                        while(col < lineWidth && col < iw && pix[row * iw + col] != whiteColor)
                       {
```

```
                            runl += scale;
                            col++;
                            totalRow += scale;
                        }
                        if(runl >= 64)
                        {
                            code = filterarrar[(int)Math.floor(runl / 64 - 1)];
                            runl -= code[2];
                            if(debug)
                            System.out.println(String.valueOf(String.valueOf(
                            (new StringBuffer("Black length=")).append(code[2]).append
                            (" code=").append(code[0]).append(" bits=").append(code[1]))));
                            setBitMask(code[0], code[1]);
                        }
                        code = whitcodearray[runl];
                        if(debug)
                         System.out.println(String.valueOf(String.valueOf(
                         (new StringBuffer("Black length=")).append(runl).append
                         (" code=").append(code[0]).append(" bits=").append(code[1]))));
                         setBitMask(code[0], code[1]);
                    }
                    if(completeLine && totalRow < lineWidth)
                    {
                        runl = lineWidth - totalRow;
                        if(runl >= 64)
                        {
                            code = matharray[(int)Math.floor(runl / 64 - 1)];
                            runl -= code[2];
                            setBitMask(code[0], code[1]);
                        }
                        code = codearray[runl];
                        setBitMask(code[0], code[1]);
                    }
                    while(minBytesLine > imagearrcount - lineBytesOffset)
                    {
                        int n = 0;
                        while(n < 8)
                        {
                            setBitMap(0);
                            n++;
                        }
                    }
                    checkBitMap();
                    if(debug)
                     System.out.println(" EOL");
                }
            }
             if(completePage)
             for(; currentPageRow < pageLength; currentPageRow++)
             {
             runl = lineWidth;
             if(runl >= 64)
             {
                code = matharray[(int)Math.floor(runl / 64 - 1)];
                runl -= code[2];
                setBitMask(code[0], code[1]);
             }
             code = codearray[runl];
             setBitMask(code[0], code[1]);
             checkBitMap();
             }
            if(createEOP)
             _$41684();
            if(leftshift != 7)
            {
                imagebit[imagearrcount] = masking;
                imagearrcount++;
            }
            byte result[] = new byte[imagearrcount];
            for(int h = 0; h < imagearrcount; h++)
            result[h] = imagebit[h];
             return result;
        }
}
```

Download this listing.

In the above listing, the ImageToFaxEncoder class encodes the fax document. The various methods defined in the ImageToFaxEncoder.java file are:

- `setBitMap()`:Sets the bitmap for the image that the Fax application sends as a fax document.
- `setBitMask()`:Sets the mask bit for the image that the Fax application sends as a fax document.
- `checkBitMap()`:Checks the bitmap for the image sent as a fax.
- `_$41684()`:Invokes the checkBitMap() method.

- `addFiller()`:Adds the file filter for the image to open only image files.
- `encodeImage()`: Encodes the image that the Fax application sends as a fax.

PREVIOUS NEXT

Converting the Encoded Fax Image into Frames

The LCFrame.java file converts the encoded image of the document sent using the Fax application into frames.

Listing 6-9 shows the LCFrame.java file:

Listing 6-9: The LCFrame.java File

```
public class LCFrame
{
    byte bytes[];
    int bytesCount;
    public LCFrame(byte b[], int len)
    {
        bytes = new byte[256];
        bytesCount = 0;
        bytes = b;
        bytesCount = len;
    }
    public LCFrame()
    {
        bytes = new byte[256];
        bytesCount = 0;
        bytes[0] = -1;
        bytesCount++;
        bytes[1] = 3;
        bytesCount++;
        bytes[2] = 0;
        bytesCount++;
    }
    public void addByte(byte b)
    {
        bytes[bytesCount++] = b;
    }
    public void setLast(boolean l)
    {
        if(l)
            bytes[1] = 19;
        else
            bytes[1] = 3;
    }
    public int getFrameType()
    {
        if(bytesCount > 2)
        {
            int b = bytes[2];
            if(b < 0)
                b = 256 + b;
            return b;
        }
      else
      {
          return 0;
      }
   }
    public void setFrameType(byte t)
    {
        if(bytesCount > 2)
            bytes[2] = t;
    }
    public boolean isLast()
    {
        if(bytesCount > 1)
            return bytes[1] == 19;
        else
            return false;
    }
    public byte[] getData()
    {
        if(bytesCount <= 5)
            return null;
        byte r[] = new byte[bytesCount - 5];
        for(int h = 0; h < r.length; h++)
            r[h] = bytes[h + 3];
        return r;
    }
    public byte[] getRawData()
    {
        byte r[] = new byte[bytesCount];
        for(int h = 0; h < r.length; h++)
            r[h] = bytes[h];
        return r;
    }
}
```

Download this listing.

In the above listing, the LCFrame class converts a fax image to frames. The various methods defined in the LCFrame.java file are:

- `addByte()`: Adds a byte to the fax document.
- `getFrameType()`: Determines the type of frame that the Fax application sends.
- `setFrameType()`: Sets the frame type for the image that the Fax application sends.
- `isLast()`:Determines whether the current data byte is the last byte or not.
- `getData()`:Retrieves the data that is converted into frames using the LCFrame class.
- `getRawData()`:Retrieves the data that the Fax application sends in the form of frames.

PREVIOUS NEXT

Sending the Fax

The SendingFax.java file sends the fax to the number specified in the Fax Number text box of the Fax Application window.

Listing 6-10 shows the SendingFax.java file:

Listing 6-10: The SendingFax.java File

```
/*Imports required java.awt.image class*/
import java.awt.Image;
/*Imports required java.i/o package classes*/
import java.io.*;
/*Imports required java.text.DateFormat class*/
import java.text.DateFormat;
/*
Imports required java.text.SimpleDateFormat class
*/
import java.text.SimpleDateFormat;
/*Imports required java.util.Calendar class*/
import java.util.Calendar;
import java.util.Date;
import javax.comm.*;
/*
class SendingFax - This class is used to send commands to modem to send fax
Constructor:
   SendingFax-This constructor creates GUI.
Methods:
   openForFax - This method is used to send fax.
actionPerformed - This method describes which action will be performed on which component.
main - This method creates the main window of the application and displays all the components.*/
class SendingFax
{
   /*Declare objects of String class.*/
      public String flowcontrolin;
      protected String modemresponse;
      public String dialingmode;
      public String noecho;
      public String ownid;
      public String logStr;
      public String initialcommands[];
      public String port;
      public String initString;
      public String faxFile;
      public String lastError;
      public String resetcommand;
      public int flowcontrol;
      public boolean bitorder;
      public boolean modemFBOR;
      public int faxclass;
      public boolean log;
      public boolean debug;
      public PreparedFaxDoc producer;
      protected int pages;
      protected ImageToFaxEncoder encoder;
      public PortFax listener;
      protected boolean V27Supported;
      protected boolean V29Supported;
      protected boolean V17Supported;
      public int MPSEOPdelay;
      public int buffersize;
      long startTime;
      long tout;
      /*
      Declare objects of CommPortIdentifier class*/
      CommPortIdentifier commportid;
      /*Declare objects of SerialPort class*/
      SerialPort serialPort;
      /*Declare objects of OutputStream class*/
      OutputStream outputStream;
      /*Declare objects of InputStream class*/
      InputStream inputStream;
      protected byte reverseBytes[];
      boolean isClass1;
      boolean isClass2;
      boolean isClass20;
      protected static double TRAINING_SECONDS = 1.5D;
      protected ModemCapabilities cap;
      public int maxRetries;
      protected static int brClass2_to_Class1[] = {24, 48, 72, 96, 121, 145};
      protected static String brClass2_to_Class1V[] = {"V27", "V27", "V29", "V29", "V17", "V17"};
       public int bitrate;
       public int resolution;
       public static final int RESOLUTION_NORMAL = 0;
       public static final int RESOLUTION_FINE = 1;
```

```
 public int timeout;
 public int resetdelay;
 public int lastresponse;
 public int pagecode;
 protected int postcode;
 public int hangcode;
 public SendingFax()
{
      reverseBytes = new byte[256];
      port = "COM1";
      isClass1 = false;
      isClass2 = false;
      isClass20 = false;
      initString = "ATV1Q0";
      maxRetries = 3;
      bitrate = 3;
      resolution = 1;
      faxFile = "tmpFax";
      lastError = "";
      resetcommand = "ATZ";
      timeout = 30;
      resetdelay = 0;
      lastresponse = 0;
      pagecode = -1;
      postcode = -1;
      hangcode = -1;
      flowcontrol = 0;
      modemresponse = "";
      noecho = "ATE0";
      flowcontrolin="";
      bitorder = false;
      modemFBOR = true;
      dialingmode = "";
      faxclass = 2;
      ownid = "12345";
      log = true;
      debug = true;
      logStr = "";
      initialcommands = new String[0];
      pages = 0;
      encoder = null;
      listener = null;
      V27Supported = true;
      V29Supported = true;
      V17Supported = false;
      MPSEOPdelay = 8;
      buffersize = 0x3d090;
      startTime = 0L;
      tout = 0L;
      /*
      Initialize object of ImageToFaxEncoder class.
      */
      encoder = new ImageToFaxEncoder();
      encoder.minBytesLine = 0;
      for(int i = 0; i < 256; i++)
     {
         byte b = (byte)i;
         byte bi = 0;
         if((b & 0x80) > 0)
             bi |= 1;
         if((b & 0x40) > 0
            bi |= 2;
         if((b & 0x20) > 0)
             bi |= 4;
         if((b & 0x10) > 0)
             bi |= 8;
         if((b & 8) > 0)
             bi |= 0x10;
         if((b & 4) > 0)
             bi |= 0x20;
         if((b & 2) > 0)
             bi |= 0x40;
         if((b & 1) > 0)
             bi |= 0x80;
         reverseBytes[i] = bi;
     }
}
/*
openForFax - This method is called to open and prepare a port to send fax.
Parameters: preparedfaxdoc - Object of PreparedFaxDoc class
Return Value:boolean
*/
public boolean openForFax(PreparedFaxDoc preparedfaxdoc)
{
/*createFaxFiles method is called.*/
     if(!createFaxFiles(preparedfaxdoc))
     {
         lastError = "Could not create fax files.";
```

```
            return false;
        }
        if(listener != null)
         listener.faxProgress(1, 0);
         /*
         Initialize object of CommPortIdentifier.
         */
        try
        {
            commportid = ommPortIdentifier.getPortIdentifier(port);
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            lastError = "Could not find port ".concat(String.valueOf(String.valueOf(port)));
            boolean flag = false;
            return flag;
    }
    if(commportid == null)
    {
        lastError = "Could not find port ".concat(String.valueOf(String.valueOf(port)));
        return false;
    }
    if(commportid.getPortType() != 1)
    {
        lastError = String.valueOf(String.valueOf(
        (new StringBuffer("")).append(port).append(" is not a serial port")));
        return false;
    }
    /*Open the serial port.*/
    try
    {
        serialPort = (SerialPort)commportid.open("Fax Application", 2000);
    }
    catch(PortInUseException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not open port ".concat(String.valueOf(String.valueOf(port)));
        boolean flag1 = false;
        return flag1;
    }
    /*Sets port receive time.*/
    try
    {
        serialPort.enableReceiveTimeout(1000);
    }
    catch(Exception exception) { }
/*Gets input and output stream.*/
    try
    {
        outputStream = serialPort.getOutputStream();
        inputStream = serialPort.getInputStream();
    }
    catch(IOException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not get streams";
        boolean flag2 = false;
        return flag2;
    }
/*Sets serial port parameter.*/
    try
    {
        serialPort.setSerialPortParams(19200, 8, 1, 0);
    }
    catch(UnsupportedCommOperationException e)
    {
        System.err.println(e.getMessage());
        lastError = "Could not configure port";
        boolean flag3 = false;
        return flag3;
    }
   if(listener != null)
   listener.faxProgress(2, 0);
   /*Send reset command to modem.*/
   if(!sendATCommand(resetcommand))
   {
        lastError = "Could not reset modem with ATZ";
        return false;
    }
  if(resetdelay > 0)
  /*softDelay method is called.*/
  softDelay(resetdelay);
  if(flowcontrol == 2 && flowcontrolin.length()>0)
  sendATCommand(flowcontrolin);
  if(flowcontrol == 1 &&   flowcontrolin.length()>0)
  sendATCommand(flowcontrolin);
  if(flowcontrol == 0 && flowcontrolin.length()>0)
```

```
sendATCommand(flowcontrolin);
try
{
   if(flowcontrol == 2)
      serialPort.setFlowControlMode(2);
      if(faxclass != 1 && flowcontrol == 1)
         serialPort.setFlowControlMode(8);
      if(flowcontrol == 0)
          serialPort.setFlowControlMode(0);
  }
  catch(Exception e)
  {
      System.err.println(e.getMessage());
  }
  /*Add flow control type in log file.*/
  if(log)
  {
      if((serialPort.getFlowControlMode() & 8) > 0) addLogLn("FLOWCONTROL_XONXOFF_OUT");
      if((serialPort.getFlowControlMode() & 4) > 0) addLogLn("FLOWCONTROL_XONXOFF_IN");
      if((serialPort.getFlowControlMode() & 1) > 0) addLogLn("FLOWCONTROL_RTSCTS_IN");
      if((serialPort.getFlowControlMode() & 2) > 0) addLogLn("FLOWCONTROL_RTSCTS_OUT");
  }
  /*Send echo command to modem.*/
  sendATCommand(noecho);
  sendATCommand(initString);
  /*queryCapabilities method is called.*/
  queryCapabilities();
  if(isClass1 && faxclass == 1)
  {
     /*
      Send service class selection command
      */
      if(!sendATCommand("AT+FCLASS=1"))
      {
          lastError = "Could not set class 1";
          return false;
      }
   /*queryBR method is called.*/
      queryBR();
  }
  if(isClass2 && faxclass == 2)
  {
      /*
      Send service class selection command
      */
      if(!sendATCommand("AT+FCLASS=2"))
      {
          lastError = "Could not set class 2";
          return false;
      }
      if(debug)
      /*Send HDLC frame tracing command.*/
      sendATCommand("AT+FBUG=1");
  }
  if(isClass20 && faxclass == 20)
  {
      /*Send service class selection command*/
      if(!sendATCommand("AT+FCLASS=2.0"))
      {
         lastError = "Could not set class 2.0";
         return false;
      }
      sendATCommand("AT+FNR=1,1,1,0");
      if(debug)
          sendATCommand("AT+FBU=1");
  }
  if(modemFBOR)
  {
      if(faxclass == 2)
      {
         if(!bitorder)
         /*
         The FBOR command determines the order in which data bits are transmitted
         between the DTE and the modem and between the modem and the PSTN.
         +FBOR=0 selects direct bit order where the first bit of a byte sent to the
         modem is the first bit sent down the PSTN.
         +FBOR=1 selects reverse bit order where the first bit of a byte sent to the
         modem is the last bit sent down the PSTN.
         */
         sendATCommand("AT+FBOR=1");
         if(bitorder)
         sendATCommand("AT+FBOR=0");
      }
      if(faxclass == 20)
      {
         /*
         Send command to setup data bit order.
         */
```

```
            if(!bitorder)
            sendATCommand("AT+FBO=1");
            if(bitorder)
            sendATCommand("AT+FBO=0");
          }
      }
      if(ownid.length() > 0)
      {
          /*Send Local ID string command.*/
          if(faxclass == 20)
          sendATCommand(String.valueOf(String.valueOf((new StringBuffer("AT+FLI=\"")).append(ownid
          if(faxclass == 2)
          /*
          This command is used to set the local identifying information stored within
          the modem. This allows the remote system to identify the calling station.
          The string may be up to 20 ASCII characters in length.
          */sendATCommand(String.valueOf(String.valueOf(
          (new StringBuffer("AT+FLID=\"")).append(ownid).append("\""))));
      }
       /*
       Send command for modem capabilities parameters.
       */
       if(faxclass == 20)
       sendATCommand(String.valueOf(String.valueOf(
       (new StringBuffer("AT+FCC=")).append(resolution).append(",").append
       (bitrate).append(",0,2,0,0,0,0"))));
       if(faxclass == 2)
       sendATCommand(String.valueOf(String.valueOf(
       (new StringBuffer("AT+FDCC=")).append(resolution).append(",").append
       (bitrate).append(",0,2,0,0,0,0"))));
       for(int i = 0; i < initialcommands.length; i++)
       {
           System.out.println("");
            sendATCommand(initialcommands[i]);
       }
       return true;
 }
/*
createFaxFiles - This method is called to create image of editpane content.
Parameters: p - Object of PreparedFaxDoc class
Return Value:boolean
*/
protected boolean createFaxFiles(PreparedFaxDoc p)
 {
   if(faxclass == 1)
         bitorder = true;
     if(p != null)
         producer = p;
     if(producer == null)
         return false;
     if(listener != null)
   /*faxProgress method is called.*/
         listener.faxProgress(6, 0);
        pages = 0;
        encoder.createEOP = faxclass == 20;
     if(faxclass == 1)
         encoder.createEOP = false;
      Image page = null;
   for(page = producer.getFaxPage(pages); page != null; page = producer.getFaxPage(pages))
   {
      byte pageBytes[] = encoder.encodeImage(page);
      if(pageBytes != null)
      {
      /*Write string in the file.*/
      writeFile(String.valueOf(String.valueOf(
      (new StringBuffer(String.valueOf(String.valueOf(faxFile)))).append(".").append
      (pages))), pageBytes, pageBytes.length);
      pages++;
      }
   }
   return true;
 }
/*
setPortName - This method is called to set the port name.
Parameters: port - Object of String class
Return Value: NA
*/
 public void setPortName(String port)
 {
  this.port = port;
 }
/*
setInitString - This method is called to set the modem initial string.
Parameters: s - Object of String class
Return Value: NA
*/
 public void setInitString(String s)
```

```
 {
  initString = s;
 }
/*
writeFile - This method is called to write string in the file.
Parameters: f - Object of String class,b[] - array of bytes,count - value of integer type.
Return Value:boolean
*/
    protected boolean writeFile(String f, byte b[], int count)
    {
     try
     {
         FileOutputStream fo = new FileOutputStream(f);
         fo.write(b, 0, count);
         fo.close();
     }
     catch(Exception e)
     {
         System.err.println("Error writing to file
         ".concat(String.valueOf(String.valueOf(e.getMessage()))));
         boolean flag = false;
         return flag;
     }
     return true;
 }
/*
sendATCommand - This method is called to send the command to the  modem.
Parameters: messageString - Object of String class
Return Value:boolean
*/
protected boolean sendATCommand(String messageString)
{
    if(messageString.trim().length() == 0)
    return true;
    if(sendATCommandnoWait(messageString))
    return waitFor("OK");
    else
    return false;
 }
/*
sendATCommandnoWait - This method is called to send the wait command to the modem.
Parameters: messageString - Object of String class
Return Value:boolean
*/
protected boolean sendATCommandnoWait(String messageString)
 {
     char cr = '\r';
     try
     {
         if(log)
         {
                addLogLn("");
                addLogLn("Command out.");
         }
         if(log)
         addLog(messageString);
         messageString = String.valueOf(messageString) + String.valueOf(cr);
         /*Write output stream.*/
         outputStream.write(messageString.getBytes());
     }
     catch(IOException e)
     {
         System.err.println(e.getMessage());
            boolean flag = false;
         return flag;
     }
     return true;
  }
   /*
   addLogLn - This method is called to display the fax status.
   Parameters: s - Object of String class
   Return Value: NA
   */
    protected void addLogLn(String s)
    {
      logStr = String.valueOf(String.valueOf(
      (new StringBuffer(String.valueOf(String.valueOf(logStr)))).append("\n").append(s)));
      if(debug)
      System.out.println(s);
    }
      /*
      softDelay - This method is called to display the fax status.
      Parameters: t - integer type variable.
      Return Value: NA
      */
    protected void softDelay(int t)
    {
```

```
     try
     {
         Thread.currentThread();
         Thread.sleep(t);
     }
     catch(Exception exception) { }
 }
/*
queryCapabilities - This method is called to set the fax class.
Parameters: NA
Return Value: Nected void queryCapabilities()
{A
*/
prot
   sendATCommand("AT+FCLASS=?");
   if(modemresponse.indexOf("1") > 0)
      isClass1 = true;
   if(modemresponse.indexOf("2") > 0)
      isClass2 = true;
   if(modemresponse.indexOf("2.0") > 0)
      isClass20 = true;
  }
   /*
   queryBR - This method is called to set the modem variables.
   Parameters: NA
   Return Value: NA
   */
 protected void queryBR()
{
   if(isClass1)
   {
      sendATCommand("AT+FTM=?");
      if(modemresponse.indexOf("48") > 0)
         V27Supported = true;
      if(modemresponse.indexOf("96") > 0)
         V29Supported = true;
      if(modemresponse.indexOf("97") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("98") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("145") > 0)
         V17Supported = true;
      if(modemresponse.indexOf("146") > 0)
         V17Supported = true;
   }
}
/*
waitFor - This method is used to invoke waitForOK method.
Parameters: resp - Object of String class.
Return Value:boolean
*/
 public boolean waitFor(String resp)
 {
   return waitForOK(resp, true);
 }
/*
waitForOK - This method is used to invoke waitForOK method.
Parameters:  resp - object of String class, wOK - boolean type variable.
Return Value:boolean
 */
protected boolean waitForOK(String resp, boolean wOK)
 {
     return waitForOK(resp, wOK, timeout);
 }
   protected void addLog(String s)
    {
        logStr = String.valueOf(logStr) + String.valueOf(s);
        if(debug)
        System.out.print(s);
    }
    /*
    waitForOK - This method is called to call waitForOK method.
    Parameters: resp - object of String class, wOK - boolean type variable, ptout - integer type
    Return Value:boolean
    */
    protected boolean waitForOK(String resp, boolean wOK, int ptout)
    {
        String r = "";
        String line = "";
        modemresponse = "";
        Calendar cal = Calendar.getInstance();
        long startTime = cal.getTime().getTime();
        try
        {
            if(log)
            {
                   addLogLn("");
                   addLog("Response: ");
```

```
    }
int c = inputStream.read();
r = String.valueOf(r) + String.valueOf((char)c);
do
{
   long now = Calendar.getInstance().getTime().getTime();
   Calendar cale = Calendar.getInstance();
   cale.add(6, 7);
   String dateStr = (new SimpleDateFormat("ddmmyyyy")).format(cale.getTime());
   if(startTime < now && now - startTime > (long)(ptout * 1000))
   {
      timeout = 9;
      lastError = "Timeout waiting for response";
      boolean flag1 = false;
      return flag1;
    }
    if(c == 13)
    {
         if(line.indexOf("+FHNG:") >= 0 || line.indexOf("+FHS:") >= 0 ||
         line.indexOf("+FHG:") >= 0)
         hangcode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FPS:") >= 0 || line.indexOf("+FPTS:") >= 0)
         pagecode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FET:") >= 0)
         postcode = (new Integer(line.substring(line.indexOf(":") + 1,
         line.length()))).intValue();
         if(line.indexOf("+FDSC:") >= 0)
        {
            String capStr = line.substring(line.indexOf(":") + 1, line.length());
            ModemCapabilities tmpCap = new ModemCapabilities();
            tmpCap.decodeCapabilitiesClass2(capStr);
         }
         if(line.indexOf("+FCON") >= 0 || line.indexOf("+FCO") >= 0)
         lastresponse = 5;
         if(line.indexOf("CONNECT") >= 0 && faxclass == 1)
         lastresponse = 5;
         if(line.indexOf("ERROR") >= 0 || line.indexOf("FCERROR") >= 0)
         {
               lastresponse = 1;
                  boolean flag2 = false;
               return flag2;
         }
         if(line.indexOf("BUSY") >= 0)
         {
               lastresponse = 8;
               lastError = "Line busy";
               boolean flag3 = false;
               return flag3;
         }
         if(line.indexOf("NO DIALTONE") >= 0)
         {
               lastresponse = 7;
               lastError = "No dial tone";
               boolean flag4 = false;
               return flag4;
          }
         if(line.indexOf("NO CARRIER") >= 0 && resp.compareTo("NO CARRIER") != 0)
         {
               lastresponse = 6;
               lastError = "No carrier";
               boolean flag5 = false;
               return flag5;
         }
          line = "";
    }
    if(log)
   addLogCh(c);
   if(c == 13 && r.indexOf(resp) >= 0 && (r.indexOf("OK") >= 0 || !wOK))
   break;
   if(c == 13 && r.indexOf("+FHNG") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag6 = false;
        return flag6;
    }
   if(c == 13 && r.indexOf("+FHG") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag7 = false;
        return flag7;
    }
   if(c == 13 && r.indexOf("+FHS") >= 0 && (r.indexOf("OK") >= 0 || !wOK))
    {
        boolean flag8 = false;
        return flag8;
    }
    if(r.indexOf("ERROR") >= 0)
    {
```

```
                    boolean flag9 = false;
                   return flag9;
                }
               c = inputStream.read();
               r = String.valueOf(r) + String.valueOf((char)c);
               modemresponse = String.valueOf(modemresponse) + String.valueOf((char)c);
               if(c != 10 && c != 13)
               line = String.valueOf(line) + String.valueOf((char)c);
            }
            while(true);
            if(log)
                addLogLn("");
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            boolean flag = false;
            return flag;
        }
        return true;
    }
/*
addLogCh - This method is called to print log document.
Parameters: c - integer type variable.
Return Value: NA
*/
protected void addLogCh(int c)
    {
        String cs = "";
        if(c < 32)
         cs = String.valueOf(String.valueOf((new StringBuffer("<")).append(c).append(">")));
        else
         cs = "".concat(String.valueOf(String.valueOf((char)c)));
         logStr = String.valueOf(logStr) + String.valueOf(cs);
        if(debug)
         System.out.print(cs);
    }
    /*
   close - This method is called to close input and output stream.
   Parameters: NA
   Return Value:boolean
   */
    public boolean close()
    {
        for(int c = 0; c < pages; c++)
        {
            File file;
            try
            {
               file = new File(String.valueOf(String.valueOf(
               (new StringBuffer(String.valueOf(String.valueOf(faxFile))))
               .append(".").append(c))));
            }
            catch(Exception exception) { }
        }
        try
        {
            inputStream.close();
            outputStream.close();
            serialPort.close();
        }
        catch(Exception e)
        {
            System.err.println(e.getMessage());
            boolean flag = false;
            return flag;
        }
        return true;
    }
/*
sendFax - This method is called to send the fax
Parameters: destId - Object of String class.
Return Value:boolean
 */
 public boolean sendFax(String destId)
    {
     hangcode = -1;
     boolean resendPage = false;
     int retry = 0;
     String r = "";
     if(listener != null)
         listener.faxProgress(3, 0);
     if(connect(destId))
     {
         if(faxclass == 1)
         {
             if(!phaseB())
                 return false;
```

```
    if(!sendTSI())
        return false;
}
int pageCount = 0;
boolean retrain = true;
System.out.println("send");
do
{
    if(pageCount >= pages)
        break;
    if(listener != null)
   listener.faxProgress(4, pageCount);
    boolean sendingUserFile = false;
    byte pageBytes[];
    if((new File("send.g3")).exists())
    {
      pageBytes = readFile("send.g3");
      sendingUserFile = true;
      System.out.println("** SENDING USER FILE *** send.g3");
    }
   else
    {
         pageBytes = readFile(String.valueOf(String.valueOf(
         (new StringBuffer(String.valueOf(String.valueOf(faxFile))))
         .append(".").append(pageCount))));
    }
    if(pageBytes == null)
    {
      lastError = "Could not read fax file.";
      return false;
    }
    if(faxclass == 1 && retrain)
    {
        if(!sendDCS())
            return false;
        if(!doTraining())
            return false;
    }
    retrain = false;
   if((faxclass == 2 || faxclass == 20) && !sendATCommandnoWait("AT+FDT"))
   {
      lastError = "Could not start phase C";
      return false;
    }
    if(faxclass == 20)
       waitForOK("CONNECT", false);
    if(faxclass == 2)
    {
         int previousFlowControl = serialPort.getFlowControlMode();
         try
           {
               serialPort.setFlowControlMode(0);
           }
            catch(Exception exception) { }
           try
           {
               if(log)
               addLog("Response: ");
               r = "";
               int c = inputStream.read();
               r = String.valueOf(r) + String.valueOf(c);
               do
               {
                   if(c == 17)
                   break;
                   c = inputStream.read();
                   if(log)
                   addLogCh(c);
                   r = String.valueOf(r) + String.valueOf(c);
                   if(c == 13 && r.indexOf("OK") >= 0)
                   break;
                   if(c == 17 && log)
                   addLogLn("XON received");
               }
               while(true);
               if(r.indexOf("+FHNG") >= 0)
               {
                  lastresponse = 2;
                  lastError = "Remote closed connection";
                  boolean flag = false;
                  return flag;
               }
               if(r.indexOf("+FHS") >= 0)
               {
                  lastresponse = 2;
                  lastError = "Remote closed connection";
                  boolean flag1 = false;
                  return flag1;
```

```
                }
            }
            catch(Exception e)
            {
                System.err.println(e.getMessage());
                lastError = "Error reading (wait for XON)";
                boolean flag2 = false;
                return flag2;
            }
            try
            {
                serialPort.setFlowControlMode(previousFlowControl);
            }
            catch(Exception exception1) { }
        }
        pagecode = -1;
        if(faxclass == 1)
        {
            if(!sendDataClass1(pageBytes, sendingUserFile))
            {
                lastError = "Could not send page bytes";
                return false;
            }
            byte eop[] = {0, 8, -128, 0, 8, -128, 0, 8, -128
            };
            sendBytes(eop, eop.length);
            byte rtc[] = {16, 3};
            sendBytes(rtc, rtc.length);
    }
    else
    if(!sendData(pageBytes))
    {
        lastError = "Could not send page bytes";
        return false;
    }
    if(faxclass == 20)
    {
        if(pageCount == pages - 1)
        {
            byte rtc[] = {16, 46};
            sendBytes(rtc, rtc.length);
        }
        else
        {
            byte rtc[] = {16, 44};
            sendBytes(rtc, rtc.length);
        }
        sendATCommand("AT+FPS?");
    }
    if(faxclass == 2)
    {
        byte rtc[] = {16, 3};
        sendBytes(rtc, rtc.length);
        try
        {
            Thread.currentThread();
            Thread.sleep(500L);
        }
        catch(Exception exception2) { }
        if(pageCount == pages - 1)
            sendATCommand("AT+FET=2");
        else
            sendATCommand("AT+FET=0");
        if(pagecode == -1)
            sendATCommand("AT+FPTS?");
    }
    resendPage = false;
    if(faxclass == 1)
    {
        if(!sendATCommand("AT+FTS=".concat(String.valueOf(String.valueOf(MPSEOPdelay)))))
            return false;
        int PPMretry = 1;
        LCFrame rsp = null;
        do
        {
            if(PPMretry > 3)
                break;
            if(!sendClass1FET(pageCount == pages - 1))
            return false;
            if(!sendATCommandnoWait("AT+FRH=3"))
            return false;
            rsp = waitForFrame(3000);
            if(rsp != null)
            break;
            lastError = "";
            PPMretry++;
        }
        while(true);
```

```
                    if(rsp == null)
                    {
                          lastError = "Response to MPS/EOP not received after 3 retries.";
                          return false;
                      }
                      if(rsp.getFrameType() == 44)
                      {
                          lastError = "Procedure interrupted";
                          return false;
                      }
                      if(rsp.getFrameType() == 172)
                      {
                          lastError = "Procedure interrupted";
                          return false;
                      }
                      if(rsp.getFrameType() == 140 && log)
                      addLogLn("Positive message confirmation.");
                      if(pageCount == pages - 1 && rsp.getFrameType() == 204)
                      {
                          retrain = true;
                          if(log)
                          addLogLn("Retrain positive.");
                      }
                      if(rsp.getFrameType() == 76)
                      {
                             retrain = true;
                          if(retry < maxRetries)
                          {
                                resendPage = true;
                                retry++;
                          }
                      }
                      if(rsp.getFrameType() == 250)
                      {
                          lastError = "Disconnect frame received";
                          hangcode = 20;
                          break;
                      }
                  }
                  if(faxclass == 2 || faxclass == 20)
                  switch(pagecode)
                  {
                     case -1:
                     case 0:
                     case 1:
                     default:
                     break;
                     case 2:
                     if(retry < maxRetries)
                     {
                         resendPage = true;
                         retry++;
                     }
                     break;
                  }
                  if(!resendPage)
                  pageCount++;
              }
              while(hangcode < 0);
              if(listener != null)
              listener.faxProgress(5, 0);
              if(faxclass == 1 && sendClass1Disconnect())
              hangcode = 0;
              hangup();
              if(hangcode == 0)
              return true;
          }
          return false;
      }
      /*
      connect - This method is called to connect the modem.
      Parameters: destId - Object of String class.
      Return Value:boolean
      */
      protected boolean connect(String destId)
       {
         String dialMode = "P";
         if(dialingmode=="Tone")
            dialMode = "T";
         if(faxclass == 1)
         {
            if(!sendATCommandnoWait(String.valueOf(String.valueOf(
            (new StringBuffer("ATD")).append(dialMode).append(destId)))))
            return false;
            waitForOK("CONNECT", false);
         }
         else
         if(!sendATCommand(String.valueOf(String.valueOf(
```

```
        (new StringBuffer("ATD")).append(dialMode).append(destId)))))
        return false;
        return lastresponse == 5;
      }
      protected boolean phaseB()
      {
          LCFrame f = null;
          boolean haveDIS = false;
          do
          {
              f = waitForFrame(10000);
              if(f == null)
                    return false;
              if(!waitForOK("OK", false))
                    return false;
              if(f.getFrameType() == 128)
              {
                     haveDIS = true;
                    cap = new ModemCapabilities();
                     byte capb[] = f.getData();
                    for(int i1 = 0; i1 < capb.length; i1++)
                     {
                         int p = capb[i1];
                         if(p < 0)
                         p = 256 + p;
                         capb[i1] = reverseBytes[p];
                     }
                    cap.decodeCapabilities(capb);
              }
              if(f.isLast())
                     break;
              sendATCommandnoWait("AT+FRH=3");
             }
              while(true);
             if(!haveDIS)
             {
              lastError = "DIS not received";
              return false;
          }
          if(!cap.t4)
         {
              lastError = "Receiver does not support T.4";
              return false;
          } else
          {
              return true;
          }
      }
     /*
     readFile - This method is called to read from file.
     Parameters: f - Object of String class.
     Return Value: byte
     */
     protected byte[] readFile(String f)
      {
          byte b[] = null;
       try
       {
           FileInputStream fi = new FileInputStream(f);
           b = new byte[(int)(new File(f)).length()];
           fi.read(b, 0, b.length);
           fi.close();
       }
       catch(Exception e)
       {
          System.err.println(String.valueOf(String.valueOf(
          (new StringBuffer("Read file ")).append(f).append(" ").append(e.getMessage()))));
           byte abyte0[] = null;
           return abyte0;
       }
       return b;
  }
/*
readFile - This method is called to test bit rate.
Parameters: NA
Return Value:boolean
*/
 protected boolean doTraining()
 {
  boolean reTry = true;
  LCFrame f = null;
  while(reTry)
  {
      sendTraining();
      reTry = false;
      sendATCommand("AT+FTS=1");
      sendATCommandnoWait("AT+FRH=3");
      do
```

```
        {
            f = waitForFrame(30000);
            if(f == null)
                return false;
        } while(!f.isLast());
        if(f.getFrameType() == 68)
        {
           lastError = "Training failed at speed ".concat(String.valueOf(String.valueOf(cap.rate)))
               reTry = true;
               int newRate2 = 1;
               if(bitrate == 5)
              newRate2 = 3;
               if(bitrate == 4)
              newRate2 = 3;
               if(bitrate == 3)
              newRate2 = 1;
               if(bitrate == 2)
              newRate2 = 1;
               if(bitrate == 1)
              newRate2 = 0;
               if(bitrate == 0)
              reTry = false;
               if(reTry)
               {
                   bitrate = newRate2;
                   lastError = "";
               }
               sendClass1FTH(3);
               if(!sendDCS())
                   return false;
              }
          }
          if(f != null && f.getFrameType() != 132)
          {
             lastError = "Confirmation to receive excepted.";
             return false;
        }
        return waitForOK("OK", false);
    }
    /*
    sendDataClass1 - This method is called to fragment the data.
    Parameters: b - array of byte type, len - variable of integer type,
    ignoreLineScan - variable of boolean type.
    Return Value:boolean
     */
       protected boolean sendDataClass1(byte b[], boolean ignoreLineScan)
        {
         int count = 0;
         int totalCount = 0;
         int len = b.length;
         byte chunk[] = new byte[buffersize];
         int bytesLine = 0;
         int minBytesLine = 0;
         int lineNumber = 0;
         String logLin = "Line: ";
         boolean error = false;
         if(!ignoreLineScan && cap.scanTime > 0)
         {
             double scanTimeSecs = (double)cap.scanTime / (double)1000;
             minBytesLine = (int)((double)((cap.rate * 100) / 8) * scanTimeSecs);
         }
         if(log)
             addLogLn(String.valueOf(String.valueOf((new StringBuffer
             ("Number of byte/line ")).append(minBytesLine).append
             (" (").append(cap.scanTime).append(" ms )"))));
            if(!sendClass1FTM(cap.rate))
                return false;
            try
            {
                if(flowcontrol == 1)
                serialPort.setFlowControlMode(8);
            }
            catch(Exception exception) { }
            try
            {
                bytesLine = 0;
                do
                {
                   if(totalCount >= b.length || error)
                     break;
                    count = 0;
                    do
                    {
                      if(count >= buffersize || totalCount >= b.length || error)
                      break;
                     byte dataByte = 0;
                     byte nextByte = 0;
                     boolean EOL = false;
```

```
                if(totalCount < b.length - 1)
                nextByte = b[totalCount + 1];
                if((b[totalCount] & 7) == 0 && nextByte == 1)
                EOL = true;
               if(bitorder)
               {
                   int p = b[totalCount++];
                  if(p < 0)
                   p = 256 + p;
                   dataByte = reverseBytes[p];
               }
               else
               {
                   dataByte = b[totalCount++];
               }
                   if(dataByte == 16)
                   chunk[count++] = 16;
                   chunk[count++] = dataByte;
                   bytesLine++;
                   if(EOL && lineNumber > 0 && bytesLine < minBytesLine)
                   for(; bytesLine < minBytesLine; bytesLine++)
                      chunk[count++] = 0;
                   if(EOL)
                   {
                       if(bitorder)
                      chunk[count++] = -128;
                         else
                         chunk[count++] = 1;
                         totalCount++;
                         lineNumber++;
                         bytesLine = 0;
                   }
             } while(true);
             if(!sendBytes(chunk, count))
                 error = true;
            } while(true);
        }
        catch(Exception e)
        {
           System.err.println(e.getMessage());
        }
        try
        {
            if(flowcontrol == 1)
            serialPort.setFlowControlMode(0);
        }
        catch(Exception exception1) { }
        return !error;
    }
   /*
   sendBytes - This method is called to send output bits.
   Parameters: b - array of byte type, len - variable of integer type.
   Return Value:boolean
   */
   protected boolean sendBytes(byte b[], int len)
   {
   try
     {
          addLog(String.valueOf(String.valueOf((new StringBuffer
          ("Send ")).append(len).append(" bytes  ... "))));
          outputStream.write(b, 0, len);
          addLogLn("OK");
     }
     catch(IOException e)
     {
          System.err.println(e.getMessage());
          boolean flag = false;
          return flag;
     }
     return true;
 }
/*
sendData - This method is called to invoke sendBytes method.
Parameters: b - array of byte type
Return Value:boolean
*/
protected boolean sendData(byte b[])
{
   int count = 0;
     int totalCount = 0;
     int len = b.length;
     byte chunk[] = new byte[65];
     try
     {
          while(totalCount < b.length)
          {
             for(count = 0; count < 64 && totalCount < b.length;)
             {
```

```
                byte dataByte = 0;
                if(bitorder)
                {
                   int p = b[totalCount++];
                   if(p < 0)
                   p = 256 + p;
                   dataByte = reverseBytes[p];
                }
                 else
                 {
                   dataByte = b[totalCount++];
                 }
                 if(dataByte == 16)
                 chunk[count++] = 16;
                 chunk[count++] = dataByte;
               }
               if(!sendBytes(chunk, count))
               {
                  boolean flag = false;
                  return flag;
             }
             if(!log);
          }
      }
      catch(Exception e)
      {
          System.err.println(e.getMessage());
      }
      return true;
  }
/*
waitForFrame - This method is called to receive frames.
Parameters: tiout - array of byte type
Return Value: LCFrame
 */
protected LCFrame waitForFrame(int tiout)
 {
      Calendar cal = Calendar.getInstance();
      long startTime = cal.getTime().getTime();
      long startFrameTime = 0L;
      String line = "";
      boolean dle = false;
      boolean started = false;
      byte bytes[] = new byte[1024];
      int byteCount = 0;
      try
      {
          if(log)
             addLog("Wait for frame: ");
          int c = inputStream.read();
       do
       {
          long now = Calendar.getInstance().getTime().getTime();
          if(now - startTime > (long)tiout)
       {
          cancelReception();
          lastresponse = 9;
          lastError = "Timeout waiting for frame";
          LCFrame hdlcframe = null;
          return hdlcframe;
       }
        if(c == 13)
        {
            if(line.indexOf("ERROR") >= 0)
            {
                lastresponse = 1;
                LCFrame hdlcframe1 = null;
                return hdlcframe1;
            }
            if(line.indexOf("FCERROR") >= 0)
            {
                lastresponse = 1;
                LCFrame hdlcframe2 = null;
                return hdlcframe2;
            }
            if(line.indexOf("OK") >= 0)
            {
                lastresponse = 10;
                LCFrame hdlcframe3 = null;
                return hdlcframe3;
            }
            line = "";
        }
        if(log)
       addLog(" ".concat(String.valueOf(String.valueOf(Integer.toHexString(c)))));
        if(c == 255 && !started)
        {
             started = true;
```

```
                    startFrameTime = cal.getTime().getTime();
                }
                if(started)
                if(!dle)
                {
                    if(c == 16)
                    dle = true;
                    else
                    bytes[byteCount++] = (byte)c;
                 }
                 else
                 {
                    if(c == 16)
                    {
                       dle = false;
                       bytes[byteCount++] = (byte)c;
                       }
                       if(c == 3)
                           break;
                   }
                    c = inputStream.read();
                     if(c != 10 && c != 13)
                    line = String.valueOf(line) + String.valueOf((char)c);
                 } while(true);
              }
              catch(Exception e)
              {
              System.err.println(e.getMessage());
              LCFrame hdlcframe4 = null;
              return hdlcframe4;
           }
           if(byteCount == 0)
               return null;
           else
            return new LCFrame(bytes, byteCount);
 }
protected boolean sendClass1Disconnect()
{
   if(!sendClass1FTH(3))
   return false;
   LCFrame DSCN = new LCFrame();
   DSCN.setLast(true);
   DSCN.setFrameType((byte)-5);
   if(!sendFrame(DSCN))
   return false;
   return waitForOK("OK", false);
}
protected boolean sendClass1FET(boolean lastPage)
{
   if(!sendClass1FTH(3, 3))
         return false;
    LCFrame FET = new LCFrame();
    FET.setLast(true);
    if(lastPage)
       FET.setFrameType((byte)47);
    else
         FET.setFrameType((byte)79);
    if(!sendFrame(FET))
         return false;
    return waitForOK("OK", false);
 }
protected boolean sendTSI()
{
 if(!sendClass1FTH(3))
      return false;
    LCFrame TSI = new LCFrame();
    TSI.setLast(false);
    TSI.setFrameType((byte)67);
    for(int i = ownid.length() - 1; i >= 0; i--)
       TSI.addByte((byte)ownid.charAt(i));
    for(int i = 0; i < 20 - ownid.length(); i++)
         TSI.addByte((byte)32);
    if(!sendFrame(TSI))
         return false;
    return waitForOK("CONNECT", false);
 }
 protected boolean sendDCS()
 {
     LCFrame DCS = new LCFrame();
     DCS.setLast(true);
     DCS.setFrameType((byte)-125);
     minCapabilities(cap);
     byte bcap[] = cap.encodeCapabilities();
     for(int i1 = 0; i1 < bcap.length; i1++)
     {
         int p = bcap[i1];
         if(p < 0)
             p = 256 + p;
```

```
            bcap[i1] = reverseBytes[p];
        }
        DCS.addByte(bcap[0]);
        DCS.addByte(bcap[1]);
        DCS.addByte(bcap[2]);
        if(!sendFrame(DCS))
            return false;
        return waitForOK("OK", false);
    }
    /*
    hangup - This method is called to hang-up the modem, killing the dial tone.
    Parameters: NA
    Return Value: NA
    */
    protected void hangup()
        {
          sendATCommand("ATH");
        }
    /*
    sendTraining - This method is called to set flow control of modem and invoke sendBytes method.
    Parameters: NA
    Return Value:boolean
    */
    protected boolean sendTraining()
    {
       /*
       Terminates transmission and waits for 7*10ms interval before responding with OK.
       ERROR is issued if the modem is on-hook.
       */
       sendATCommand("AT+FTS=7");
       if(!sendClass1FTM(cap.rate))
          return false;
            try
            {
                if(flowcontrol == 1)
                serialPort.setFlowControlMode(8);
            }
            catch(Exception exception) { }
            byte b[] = new byte[(int)((double)((cap.rate * 100) / 8) * TRAINING_SECONDS) + 1];
            for(int i = 0; i < b.length; i++)
              b[i] = 0;
          b[b.length - 1] = 1;
          sendBytes(b, b.length);
          byte b1[] = {16, 3};
          sendBytes(b1, b1.length);
          try
          {
              if(flowcontrol == 1)
             serialPort.setFlowControlMode(0);
            }
            catch(Exception exception1) { }
            return waitFor("OK");
        }
       protected boolean sendClass1FTH(int rate)
        {
          return sendClass1FTH(rate, timeout);
        }
       /*
       sendClass1FTM - This method is called to set transmits data using
       HDLC protocol and the defined modulation.
       Parameters: rate - Variable of integer type, pout - variable of integer type.
       Return Value:boolean
       */
       protected boolean sendClass1FTH(int rate, int pout)
       {
          String fth = "AT+FTH=".concat(String.valueOf(String.valueOf(rate)));
          int retry = 0;
          do
          {
             if(retry >= 3)
             break;
             if(!sendATCommandnoWait(fth))
             return false;
             if(waitForOK("CONNECT", false, pout))
             break;
             retry++;
          }
          while(true);
          return retry < 3;
    }
    /*
    sendClass1FTM - This method is called to set transmits data according to the defined modulation
    Parameters: rate - Variable of integer type.
    Return Value:boolean
    */
    protected boolean sendClass1FTM(int rate)
        {
          String ftm = "AT+FTM=".concat(String.valueOf(String.valueOf(rate)));
```

```
          if(!sendATCommandnoWait(ftm))
          return false;
          return waitForOK("CONNECT", false);
        }
        protected void cancelReception()
        {
         if(log)
             addLogLn("Cancel: ");
            byte cancelByte[] = {24};
            sendBytes(cancelByte, 1);
            waitFor("OK");
     }
    /*
    sendFrame - This method is called to send frames.
    Parameters: f - object of LCFrame class.
    Return Value:boolean
    */
    protected boolean sendFrame(LCFrame f)
     {
         byte b[] = f.getRawData();
         if(log)
         {
             addLogLn("");
             addLog("Send frame: ");
             for(int i = 0; i < b.length; i++)
             addLog(" ".concat(String.valueOf(String.valueOf(Integer.toHexString(b[i] & 0xff)))));
          }
          startTimer(2500);
          if(!sendBytes(b, b.length))
                      return false;
                    byte endOfFrame[] = {16, 3};
                    if(!sendBytes(endOfFrame, 2))
                     return false;
                    if(timeout())
                    {
                        lastError = "timeout sending frame.";
                        return false;
                    }
                    else
                    {
                        return true;
                    }
                 }
                /*
                minCapabilities - This method is called to set modem capabilities.
                Parameters: cap - object of ModemCapabilities class.
                Return Value: NA
                */
                protected void minCapabilities(ModemCapabilities cap)
                {
                   int desiredRate = brClass2_to_Class1[bitrate];
                  if(desiredRate < cap.rate)
                  {
                      cap.rate = desiredRate;
                      cap.vRate = brClass2_to_Class1V[bitrate];
                  }
                  if(cap.resolution == 1 && resolution == 0)
                      cap.resolution = 0;
                  if(encoder.lineWidth < cap.width)
                      cap.width = encoder.lineWidth;
                  cap.huffman = 1;
              }
             /*
             startTimer - This method is called to set starting time.
             Parameters: t - Integer type variable.
             Return Value: NA
             */
             protected void startTimer(int t)
             {
               tout = t;
               Calendar cal = Calendar.getInstance();
               startTime = cal.getTime().getTime();
             }
             /*
             timeout - This method is called to check time out.
             Parameters: NA
             Return Value:boolean
             */
             protected boolean timeout()
             {
                long now = Calendar.getInstance().getTime().getTime();
                return now - startTime > tout * (long)1000;
              }
}
```

Download this listing.

In the above listing, the SendingFax class sends a fax to the fax number of the destination. The various methods defined in the listing are:

- `addLogLn()`:dds a line that depicts the status of the Fax application in a log file.
- `addLogCh()`:Adds a character that depicts the status of the modem to the log file.
- `readFile()`:Reads the log file that contains status information about the modem.
- `openForFax()`:Opens the selected port to send the fax.
- `createFaxFiles()`:Creates an image for the file that is open in the edit pane of the Fax Application window.
- `setPortName()`:Sets the name of the port that sends the fax.
- `setInitString()`:Sets the initial string for the modem.
- `writeFile()`:Writes the file that is open in the edit pane of the Fax Application window to a temporary file on a physical drive. The createFaxFiles() method invokes the writeFile() method to write the file.
- `sendATCommand()`:Sends the AT command to the modem. The AT command checks if the modem is ready to send the fax.
- `sendATCommandnoWait()`:Sends the wait command to the modem.
- `softDelay()`: Suspends the processing of the main thread for a specified period to display the status of the Fax application. The end user specifies the duration of the delay.
- `queryCapabilities()`:Retrieves the encoding and decoding capabilities of the modem.
- `waitFor()`:Invokes the waitForOK() method.
- `waitForOK()`:Waits for the OK command that the modem sends if the modem is ready for communication.
- `close()`:Closes the input and output streams of the Fax application.
- `sendFax()`:Sends the fax document to the recipient number specified in the Fax Number text box of the Fax Application window.
- `connect()`:Connects to the modem.
- `sendDataClass1()`:Sends the fax document if the modem used is of Class 1 type. The three different classes of modems are Class 1, Class 2, and Class 2.0.
- `sendBytes()`:Helps the modem send the fax document in the form of bytes to the recipient.
- `sendData()`:Invokes the sendBytes() method to send the fax document.
- `waitForFrame()`:Sets the delay time between two frames.
- `sendClass1Disconnect()`: Disconnects a Class 1 modem.
- `sendClass1FET()`:Sets the number of bytes in a frame.
- `sendTSI()`:Sends the frames of the fax document to the Transmit Station ID (TSI) modem and waits for a response from the modem.
- `sendDCS()`:Determines the type of modem, and sends the frames of the fax document to the modem if the modem type is Digital Command Signal (DCS).
- `hangup()`:Disconnects the modem.
- `sendTraining()`:Sets the flow control property of the modem and invokes the sendBytes() method.
- `sendClass1FTH()`:Transmits data using the High-Level Data Link Control (HDLC) protocol and the defined modulation.
- `sendClass1FTM()`:Transmits data according to the defined modulation.
- `cancelReception()`:Sends the cancel command to the modem.
- `sendFrame()`:Sends the frames of the fax document to the recipient.
- `minCapabilities()`:Retrieves the minimum encoding and decoding capabilities of the modem.
- `startTimer()`:Starts a timer for the Fax application.
- `timeout()`:Closes the connection if the recipient does not respond for a specific period.

Unit Testing

To test the Fax application:

1. Download the Javacomm20-win32 JCA, which is available in a zip format at:

   http://java.sun.com/products/javacomm/downloads/index.html

2. Unzip the zip file downloaded from the above URL.
3. Copy win32com.dll from the unzipped file to the jre\bin directory where J2SDK is installed.
4. Copy comm.jar from the unzipped file to the jre\lib\ext directory where J2SDK is installed.
5. Copy javax.comm.properties from the unzipped file to the jre\lib directory where J2SDK is installed.
6. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:
   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```
7. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:
   ```
   set classpath = %classpath%;d:\j2sdk1.4.0_02\lib; d:\j2sdk1.4.0_02\jre\lib\ext\comm.jar
   ```
8. Copy the Java files for the Fax application to a folder on your computer. Use the cd command at the command prompt to move to that folder. Compile the files using the javac command, as follows:
   ```
   javac *.java
   ```
9. Run the Fax application using the following command at the command prompt:
   ```
   java PortFax
   ```
10. Select File->Open to open the document that you want to send as a fax using the Fax application. The selected file opens in the edit pane of the Fax Application window, as shown in Figure 6-4:

Figure 6-4: The Fax Application Window

11. Select the HTML check box if you want to send the fax document as an HTML page.
12. Specify the fax number in the Fax Number text box.
13. Select File->View Properties to set the properties of the port to which the modem is connected.
14. Click the Send Fax button to send the document as a fax document.
15. Click the Close button to close the Fax application.

Index

A-J

Index

M-U

List of Figures

List of Tables

Chapter 1: Introducing the Java Communication API

List of Listings

Chapter 2: Creating a Port Information Application

Chapter 3: Creating the Modem Dialer Application

Chapter 4: Creating a Printing Application

Chapter 5: Creating a Serial Communication Application

Chapter 6: Creating a Fax Application

Java InstantCode: Developing Applications Using JCA

SkillSoft Press © 2004

This book describes how JCA defines a development methodology and suggests analysis and design patterns that are useful both for building connectors to legacy applications and for designing adapters for new applications.

Table of Contents