SOFTWARE DEVELOPED TO CAPTURE AND PLAYBACK VIDEO FROM A VIDEO CAMERA VIA USB CABLE.

ATTA MUHAMMAD
2000\9806EE

ELECTRICAL AND COMPUTER ENGINEERING

OCTOBER, 2006

SOFTWARE DEVELOPED TO CAPTURE AND PLAYBACK VIDEO FROM A VIDEO CAMERA VIA USB CABLE.

ATTA MUHAMMAD
2000\9806EE

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF BACHELOR OF ENGINEERING (B.ENG) IN ELECTRICAL AND COMPUTER ENGINEERING SCHOOL OF ENGINEERING AND ENGINEERING TECHNOLOGY

FEDERAL UNIVERSITY OF TECHNOLOGY MINNA, NIGER STATE

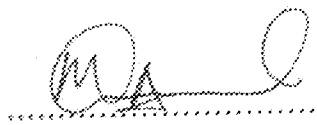
OCTOBER, 2006

DEDICATION

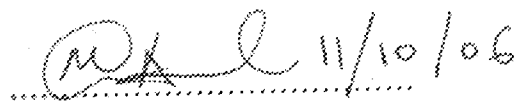
I dedicate this to my family.

DECLARATION

I Atta Muhammad, Declare that this work was done by me and has never been presented elsewhere for the award of a degree.



Atta Muhammad



Signature and Date

Engr.M.D. Abdullahi

Signature and Date

Mr. S.N. Ranala

Signature and Date

External Supervisor

Signature and Date

ACKNOWLEDGEMENT

I send my gratitude to Almighty Allah, The Author and Dispenser of Knowledge and wisdom for giving me the power to write this project. Thanks to my Mom whom has not only supported me through school but has supported me all my life. Muhammad Ozigis. Bashir Muhammad, Aminu Kandi, Aminu Lawal, Imam, and thanks to Ifeanyi, couldn't have done this without him.

Thanks to my supervisor, Mr. S.N. Romala for his time and effort

ABSTRACT

A program developed to capture video and picture transmitted from a USB type video camera and stores the received data to designated memory. The software was written in java with the Java Media Framework (JMF) application programming interface (API) incorporated to synchronize and control time based data.

TABLE OF CONTENTS

List of Figures

List of tables

CHAPTER ONE

Introduction

1.1 Project overview

Security is some what of a necessity and as the world is relying on solutions provided by technology, previous methods are being set aside. Virtually all aspects of life have been touched by technology if not all. This project focuses on the development of a program that stores captured video and still images as demanded by the user and play them back as required from a camera via the universal serial bus port or USB. The program makes up the software component of a security system. The idea of creating a program for video capture and playback was conceived as a result of the relocation of the school of engineering and engineering technology to the permanent site of the federal university. Having in place effective measures of security at the permanent site will increase the safety and more importantly the subconscious feeling that comes from being in a secure environment of the people living, working or just giving and getting lectures over there. Prevention is better than cure, having a system in place to safe guard or watch over lives and property should count as a plus, although in some cases may not function as such but serve as a deterrent.

The program was adapted from the original idea of a security system software to a more functional capture and playback program which could serve various purposes. Other than security purposes it could also provide spot on observation for instance in laboratory experiments where a specimen needs to be observed round the clock for physically visible changes. With the right hardware in place to work with this software, a system could be rigged to function as the observer and then storing images for later analysis. Businesses can

monitor and record activities in offices, hallways and warehouses. Home owners can monitor every thing from the baby's room to the backyard. Also in a close circuit television system it can provide the software requirement needed to bridge the hardware and the systems user. In traffic monitoring many cities and motorway networks have extensive traffic monitoring systems, using close circuit television to detect congestion and notice accidents. A more recent development of close circuit television is close circuit digital photography. In the instance of the necessity for clarity from the captured images, close circuit digital photography prevails over close circuit television because of the use of mega pixel digital still cameras that can take 1600 x 1200 pixel resolution images or more of the camera scene depending on the camera in use.

The program which was written in java was designed to be small, simple, and portable across platforms and operating systems although Windows is the primary OS it has been designed for and designed with.

1.2 Aim and objective.

To create a program that will capture and playback video signals transmitted from video camera to an IBM compatible computer running windows 98/me/2000/XP via a USB 2.0 cable.

1.3 Research methodology.

Primary and secondary sources of data amplified research and development. The primary source of data involved interviews with people related to some portion of this project and

personal observation. For secondary data, text books, journals, websites and magazines where engaged.

1.4 Software Overview.

The software was designed to interface a video camera. Running the software on the Windows Platform, the camera in use will require the video for Windows (VFW) driver. If it where to run on the Linux platform, Video for Linux will be required. With a camera connected via the Universal Serial Bus (USB) port of the system, the software makes it possible for the user to view video conveyed from the camera givig the user alternative choices depending on the number of cameras attached. It also provides the facility to capture live video from the camera and save the captured video to a file in a video format supported by most video players . The video formats supported are AVI(Audio-Video Interleaved), MOV (Quicktime) and MPG (MPEG-1 Video). It offers video size attribute settings which is measured in pixels, also the video format with respect to color information (RGB,YUV,CInepak) and audio formats for cameras attached that support audio capture. The software allows pictures to be taken and saved.

It incorporates a video/player that was included to provide a means of playing back captured video and videos that it supports even if they are not sourced from this software.

1.5 Camera specification and operating platform/system issues.

The video camera used to carry out this project is a zenith web cam which matches the specifications listed above. It is a low resolution camera with its highest resolution at 640X

480 pixels and supports the video for Windows driver. The cameras resolution was a significant constraint during the development of this program.

The operating platform/system used for this project is the Microsoft Windows XP operating system (98, 2000, NT, Milenium edition and both XP service packs can run the program).

The software was developed specifically with the following platforms.

1. The Java Media Framework (JMF) versione2.1.1e
2. The java development kit (JDK) versions 5.0 update 6.
3. Notepad ++ served as the code editor. The source code was typed and edited within Notepad.
4. Microsoft Windows XP service pack 2 was the operating system installed on the systems used to develop the software.
5. Exe 4J from EJ Technologies is a program designed to create executable files. It was used to create the .EXE file for this program.
6. Winrar was used for creating and packaging the installer.

1.6 Terms Defined.

- **Software.**

The term software refers to all electronic instructions in the computers memory. this is to say that without these instructions man cannot put any automated hardware to use. From this statement, it is recalled that the computer system is only complete if these three things - hardware. Software and human ware are available.

In the science of computer, the term software is categorized into two; we have system software and application software. While the former is responsible for

Directing the operation of the computer, the later is installed in the computer to enable man solve his problem. An example of system software is the operating system which is a program that directs the operation of the system as well as serving as interface between the system and the system user.

• Universal Serial Bus (USB)

USB is an open interface standard that has been included on PC motherboards since late 1997. It is an external bus that supports Plug and Play installation. Using USB, you can connect and disconnect devices without shutting down or restarting your computer. You can use a single USB port to connect up to 127 peripheral devices (which would mean 127 cameras in relation to this project). The first versions of the USB spec (1.0 and 1.1) supported maximum transfer rates of 1Mbyte/sec. version 2.0 of the USB specification supports transfer rates of 480Mbits/sec [1].

• Media Capture

Time-based media captured from a live source for processing and playback. For example, audio can be captured from a microphone or a video capture card can be used to obtain video from a camera.

A capture device might deliver multiple media streams. For example, a video camera might deliver both audio and video.

CHAPTER TWO

Literature review

2.1 Introduction

A clear understanding of the Java Media Framework (JMF) will shed light on the theoretical background of this project. The following sections walk through the various aspects of Java Media Framework (JMF) with a bid to give details on the API.

2.2 The Java Media Framework

The Java Media Framework (JMF) is an application programming interface (API) for incorporating time-based media into Java applications and applets. The Java Media Framework can be used to develop full Media systems which can play Videos, Audios, view and capture from a video camera, also capture from an audio device like microphones, perform RTP (Real-Time transfer Protocol) operations. JMF is also used to develop browser based web conferencing applications [2].

The JMF API was initially released in 1998. The first API which was JMF 1.0 API enabled programmers to develop Java programs that presented time-based media only. With the advent of the JMF 2.0 API, the framework was extended to provide support for capturing and storing media data, controlling the type of processing that is performed during playback, and performing custom processing on media data streams. In addition, JMF 2.0 defines a plug-in API that enables advanced developers and technology providers to more easily customize and extend JMF functionality. A plug-in is a program or a kind of Sub-API that enables programmers to increase the capabilities or the level of functionality of the programs.

It is usually installed into a pre-existing Application or API (API stands for Application Programming Interface). The version of JMF used for our project is JMF 2.1.1e API. To use JMF, you won't find it difficult to meet the hardware and software requirements. Your old 166-MHz Pentium proves sufficient as long as it has at least 32 MB of RAM.

2.3 Design Goals for the JMF API

The JMF API supports media capture and addresses the needs of application developers who want additional control over media processing and rendering. It also provides a plug-in architecture that provides direct access to media data and enables JMF to be more easily customized and extended. JMF is designed to:

1. Be easy to program
2. Support capturing media data.
3. Enable the development of media streaming and conferencing applications in Java.
4. Enable advanced developers and technology providers to implement custom solutions based on the existing API and easily integrate new features with the existing framework.
5. Provide access to raw media data.
6. Enable the development of custom, downloadable demultiplexers, codecs, effects processors, multiplexers, and renderers (JMF plug-ins).
7. Maintain compatibility with other versions of JMF.

Theoretical Background

2.4 Working with Time-Based Media

Any data that changes meaningfully with respect to time can be characterized as time-based media. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media. Such media data can be obtained from a variety of sources, such as local or network files, cameras, microphones, and live broadcasts.

- **Streaming Media**

Streaming Media or time based media, is a term used to describe any flow of data that requires reception and processing of data in real time. It is basically a steady stream of information that needs to be addressed, processed and presented on the fly in order to either present it to the user or record it into a file [3]

- **Content Type**

The format in which the media data is stored is referred to as its content type. QuickTime, MPEG, and WAV are all examples of content types. Content type is essentially synonymous with file type-content type is used because media data is often acquired from sources other than local files.

- **Media Streams**

A media stream is the media data obtained from a local file, acquired over the network, or captured from a camera or microphone. Media streams often contain multiple channels of data called tracks. For example, a QuickTime file might contain both an audio track and a

video track. Media streams that contain multiple tracks are often referred to as multiplexed or complex media streams.

Demultiplexing is the process of extracting individual tracks from a complex media stream.

A track's type identifies the kind of data it contains, such as audio or video. The format of a track defines how the data for the track is structured.

A media stream can be identified by its location and the protocol used to access it. For example, a URL might be used to describe the location of a QuickTime file on a local or remote system. If the file is local, it can be accessed through the FILE protocol. On the other hand, if it's on a web server, the file can be accessed through the HTTP protocol. A media locator provides a way to identify the location of a media stream when a URL can't be used. Media streams can be categorized according to how the data is delivered:

1. Pull--data transfer is initiated and controlled from the client side. For example, Hypertext Transfer Protocol (HTTP) and FILE are pull protocols.
2. Push--the server initiates data transfer and controls the flow of data. For example, Real-time Transport Protocol (RTP) is a push protocol used for streaming media. Similarly, the SGI MediaBase protocol is a push protocol used for video-on-demand (VOD).

• Common Media Formats

The following tables identify some of the characteristics of common media formats. When selecting a format, it's important to take into account the characteristics of the format, the target environment, and the expectations of the intended audience. For example, if you're

delivering media content via the web, you need to pay special attention to the bandwidth requirements.

The CPU Requirements column characterizes the processing power necessary for optimal presentation of the specified format. The Bandwidth Requirements column characterizes the transmission speeds necessary to send or receive data quickly enough for optimal presentation.

Table 2.3 5.0 Common video formats.

Format	Content Type	Quality	CPU Requirements	Bandwidth Requirements
Cinepak	AVI QuickTime	Medium	Low	High
MPEG-1	MPEG	High	High	High
H.261	AVI RTP	Low	Medium	Medium
H.263	QuickTime AVI RTP	Medium	Medium	Low
JPEG	QuickTime AVI RTP	High	High	High
Indeo	QuickTime AVI	Medium	Medium	Medium

Table 2.3.5.1 Common audio formats.

Format	Content Type	Quality	CPU Requirements	Bandwidth Requirements
PCM	AVI QuickTime WAV	High	Low	High
Mu-Law	AVI QuickTime WAV RTP	Low	Low	High
ADPCM (DVI, IMA4)	AVI QuickTime WAV RTP	Medium	Medium	Medium
MPEG-1	MPEG	High	High	High
MPEG Layer3	MPEG	High	High	Medium
GSM	WAV RTP	Low	Low	Low
G.723.1	WAV RTP	Medium	Medium	Low

Some formats are designed with particular applications and requirements in mind. High-quality, high-bandwidth formats are generally targeted toward CD-ROM or local storage applications. H.261 and H.263 are generally used for video conferencing applications and are optimized for video where there's not a lot of action. Similarly, G.723 is typically used to produce low bit-rate speech for telephony applications.

• Media Presentation

Most time-based media is audio or video data that can be presented through output devices such as speakers and monitors. Such devices are the most common destination for media data output. Media streams can also be sent to other destinations--for example, saved to a file or transpitted across the network. An output destination for media data is sometimes referred to as a data sink.

• Presentation Controls

While a media stream is being presented, VCR-style presentation controls are often provided to enable the user to control playback. For example, a conurol panel for a movie player might offer buttons for stopping, starting, fast-forwarding, and rewinding the movie.

• Presentation Quality

The quality of the presentation of a media stream depends on several factors, including:

1. The compression scheme used
2. The processing capability of the playback system
3. The bandwidth available (for media streams acquired over the network)

Traditionally, the higher the quality, the larger the file size and the greater the processing power and bandwidth required. Bandwidth is usually represented as the number of bits that are transmitted in a certain period of time--the bit rate.

To achieve high-quality video presentations, the number of frames displayed in each period of time (the frame rate) should be as high as possible. Usually movies at a frame rate of 30

frames-per-second are considered indistinguishable from regular TV broadcasts or video tapes.

* **Media Processing**

A media clip must be processed before it is played. To process a media clip the program must access a media source, create a Controller for that source and output the media [4].

In most instances, the data in a media stream is manipulated before it is presented to the user. Generally, a series of processing operations occur before presentation:

1. If the stream is multiplexed, the individual tracks are extracted.
2. If the individual tracks are compressed, they are decoded.
3. If necessary, the tracks are converted to a different format.
4. Effect filters are applied to the decoded tracks (if desired).

The tracks are then delivered to the appropriate output device. If the media stream is to be stored instead of rendered to an output device, the processing stages might differ slightly. For example, if you wanted to capture audio and video from a video camera, process the data, and save it to a file:

1. The audio and video tracks would be captured.
2. Effect filters would be applied to the raw tracks (if desired).
3. The individual tracks would be encoded.
4. The compressed tracks would be multiplexed into a single media stream.
5. The multiplexed media stream would then be saved to a file.

Demultiplexers and Multiplexers

A demultiplexer extracts individual tracks of media data from a multiplexed media stream. A multiplexer performs the opposite function, it takes individual tracks of media data and merges them into a single multiplexed media stream.

2.5 Capture Devices

To capture time-based media you need specialized hardware--for example, to capture audio from a live source, you need a microphone and an appropriate audio card. Similarly, capturing a TV broadcast requires a TV tuner and an appropriate video capture card. Most systems provide a query mechanism to find out what capture devices are available.

Capture devices can be characterized as either push or pull sources. For example, a still camera is a pull source--the user controls when to capture an image. A microphone is a push source--the live source continuously provides a stream of audio.

The format of a captured media stream depends on the processing performed by the capture device. Some devices do very little processing and deliver raw, uncompressed data. Other capture devices might deliver the data in a compressed format.

- **Capture Controls**

Controls are sometimes provided to enable the user to manage the capture process. For example, a capture control panel might enable the user to specify the data rate and encoding type for the captured stream and start and stop the capture process.

JavaTM Media Framework (JMF) provides a unified architecture and messaging protocol for managing the acquisition, processing, and delivery of time-based media data. JMF is designed to support most standard media content types, such as AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, and WAV.

By exploiting the advantages of the Java platform, JMF delivers the promise of "Write Once, Run AnywhereTM" to developers who want to use media such as audio and video in their Java programs. JMF provides a common cross-platform Java API for accessing underlying media frameworks. JMF implementations can leverage the capabilities of the underlying operating system, while developers can easily create portable Java programs that feature time-based media by writing to the JMF API.

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

2.6 High-Level Architecture

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

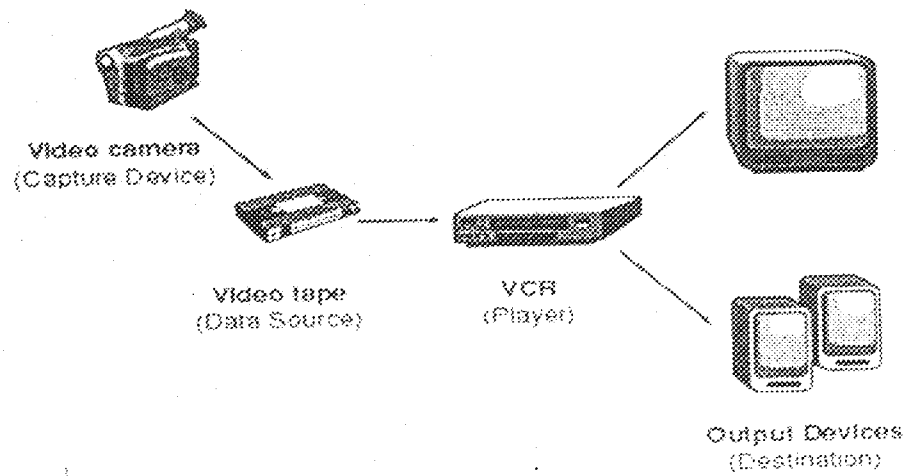


Fig.2.6 Recording, processing, and presenting time-based media.

JMF uses this same basic model. A data source encapsulates the media stream much like a video tape and a player provides processing and control mechanisms similar to a VCR. Playing and capturing audio and video with JMF requires the appropriate input and output devices such as microphones, cameras, speakers, and monitors.

Data sources and players are integral parts of JMF's high-level API for managing the capture, presentation, and processing of time-based media. JMF also provides a lower-level API that supports the seamless integration of custom processing components and extensions. This layering provides Java developers with an easy-to-use API for incorporating time-based media into Java programs while maintaining the flexibility and extensibility required to support advanced media applications and future media technologies.

• Controls

JMF Control provides a mechanism for setting and querying attributes of an object. A Control often provides access to a corresponding user interface component that enables user

control over an object's attributes. Many JMF objects expose Conurols, including Controller objects, DataSource objects, DataSink objects, and JMF plug-ins.

Any JMF object that wants to provide access to its corresponding Control objects can implement the Controls interface. Controls define methods for retrieving associated Control objects. DataSource and PlugIn use the Controls interface to provide access to their Control objects.

• User Interface Components

A Control can provide access to a user interface Component that exposes its control behaviour to the end user. To get the default user interface component for a particular Control, you call getControlComponent. This method returns an AWT Component that you can add to your applet's presentation space or application window.

A Controller might also provide access to user interface Component. [5]

2.7 Previous works by others

Cam Wizard- Is CCTV software, turns any PC and webcam into a full motion detection, CCTV camera & VCR system. . Cam Wizard detects motion and starts recording into a WMV file (Windows Media Video). The WMV movie can also be sent to any email address. Cam Wizard is compatible with all USB web cams. A full year of CCTV web cam surveillance video capture can fit onto a 100GB HD. With loud audio alarm siren function. Also great for covert CCTV surveillance.

EasyCCTV- captures images up to 30 frames per second from any USB video device. The program performs simultaneous recording. Your camera(s) can be used as part of a surveillance system. When the program detects motion in the monitored area, it sounds an alarm, e-mails you captured images, or records video. The application is password protected and may be used as home alarm.

River Past Cam Do- is an easy-to-use and fast webcam recording software. Pick the webcam from the list and record to AVI, DivX, MKV, OGM, WMV, Xvid, or MOV, MP4 or 3GP/3GP2 (Optional). Pick your video codec and quality setting. Option to record audio from any sources, including microphone, line-in, or speakers. It has a built-in scheduler for unattended recording. You can set the start time and/or the stop time.

Visual Hindsight Home Edition Monitor your home or business over the Internet with Visual Hindsight--powerful, simple to use, affordable to own! Visual Hindsight is the ideal remote network camera surveillance and recording tool for private or commercial use. Perfect for remote monitoring of convenience stores, gas stations, day care centres, retirement facilities, hospitals, laboratories, commercial property, private homes, and more.

Open Video Capturer- Open Video Capture can capture video from webcam, TV Tuner card, digital video, digital camera and other capture devices. It can set video compression codec and audio compression codec, output frame size and frame rate. It also snapshots pictures with the hotkey. It can set each capture device's properties, preview and record real-

time video, watch TV if TV Tuner is installed. It encodes AVI files with DIVX, XVID, DV Video Encoder, etc.

WebCam Monitor- WebCam Monitor turns your PC into a video surveillance system that allows you to monitor your home/office from a remote location.

WebCam Monitor can be configured to manually or automatically capture snapshot images, or record video and audio images. When an alarm condition is detected, the program can flash your computer screen, sound an audible alarm, or send you an email. **TinCam-** TinCam is a powerful webcam program that will update your homepage with webcam pictures or a live video stream. TinCam will also create a webcam homepage for you so you can get you live webcam running in no time. It is easy to use and has many great features: Easily creates a webcam homepage, Interval updates, Scheduled updates, Motion detection updates, live video streaming, Supports multiple cameras, advanced captions, Overlay image, pictures before upload, Records .AVI videos with compression, Setup Wizard for easy and fast setup. Starts with windows and runs unnoticed in system tray.

CHAPTER THREE

Design and Implementation

3.1 Introduction

This section of the project deals with the design of the software which is adequately depicted with the flow charts shown

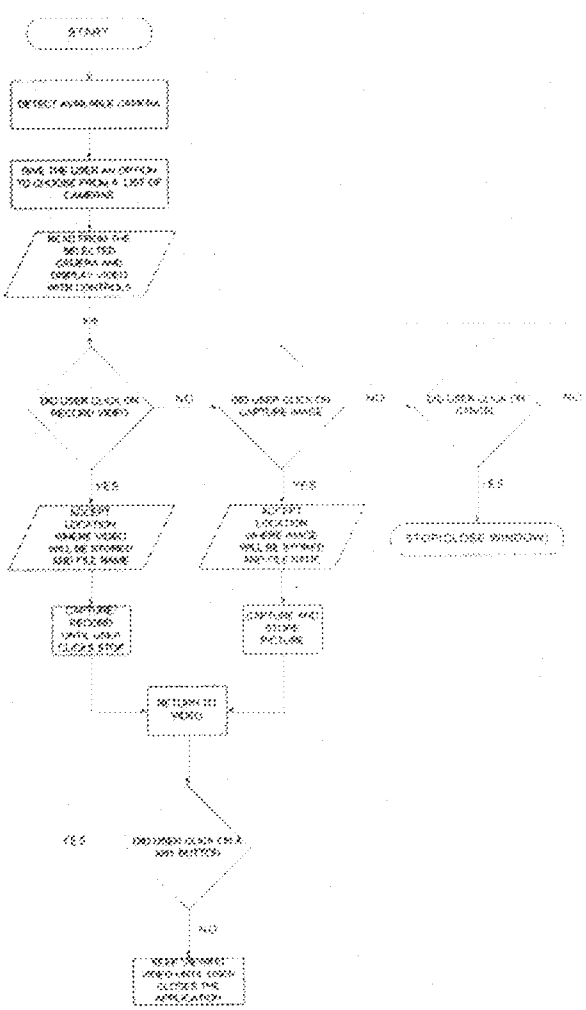


Fig.3.0 Viewing and recording videos\ capturing images

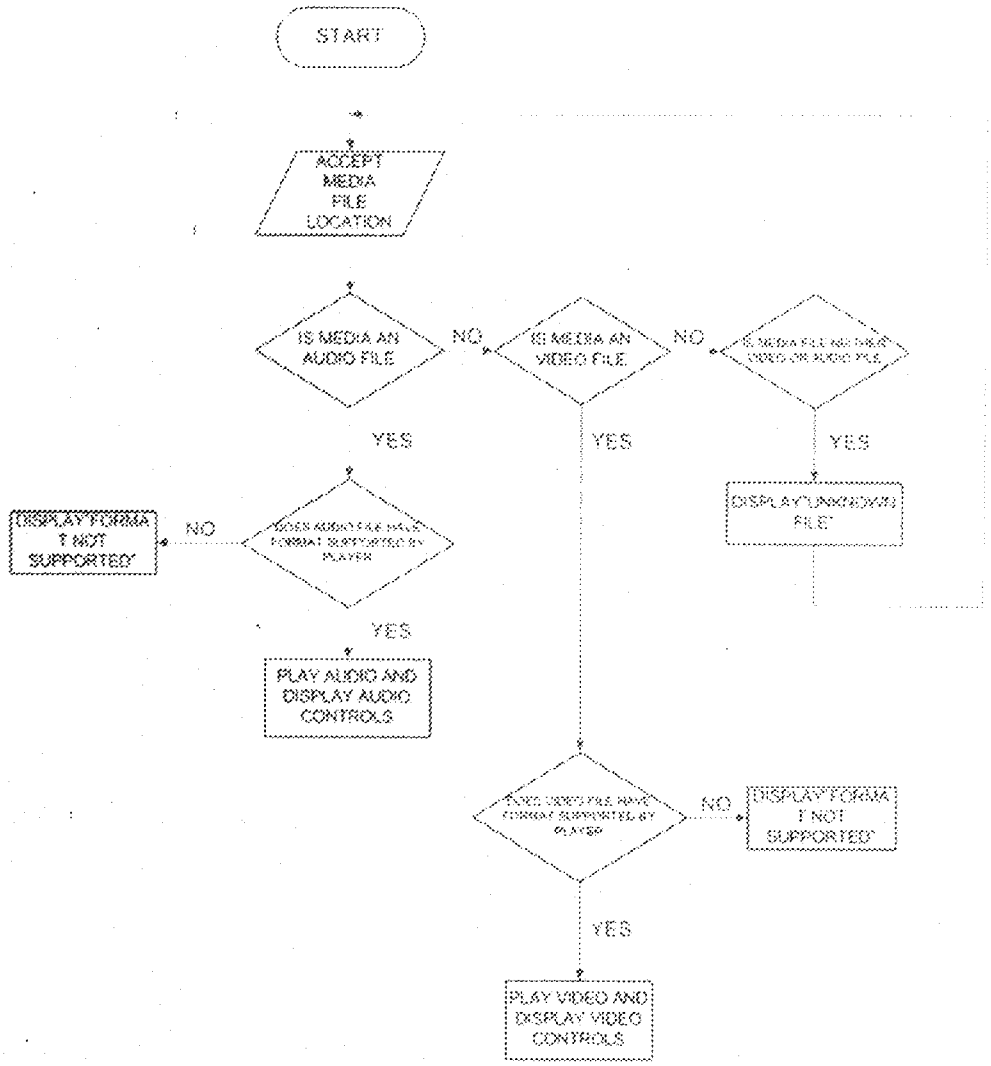


Fig.3.1 Playing video and audio

To create a program in Java, the steps are as follows

1. **Create a source file**-A source file contains text, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files.

2. **Compile the source file into a .class file.**-The Java compiler, javac, takes your source file and translates its text into instructions that the Java Virtual Machine can understand. The instructions contained within this file are known as bytecodes. Compiling the source file entailed bringing up a shell, or "command," window (typing cmd in the run window). The prompt shows the current directory. The current directory will have to be changed to the directory where your file is located. At the prompt, type the command javac *name of App*.java and press Enter where the text in italics should contain the name source file. If your prompt reappears without error messages the program was successfully compiled. The compiler has generated a bytecode file, *name of App*.class.
3. **Run the program.**-The Java launcher (java) uses the Java Virtual Machine to run your application.

   In the same directory, enter at the prompt java then a space *name of App* .

3.2 CODE LISTINGS

The codes generated to execute this project are listed below.

- **GetCapture DeviceInfo.java**

```
//importing/including external libraries and code files
-------------------------------
import javax.swing.*;
import javax.swing.JOptionPane.*;
import javax.swing.border.*;
import javax.swing.BorderFactory.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.awt.Component;
import javax.swing.JOptionPane;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.format.VideoFormat;
import javax.media.format.AudioFormat;
import java.awt.event.*;
import javax.swing.event.*;
import java.net.MalformedURLException;
```

```
import javax.swing.JFileChooser;
import javax.media.control.*;
import javax.media.Manager.*;
import javax.media.format.*;
import java.awt.image.*;
import javax.imagein.*;
import javax.media.util.BufferToImage.*;
import javax.media.util.*;
-----------------------------------------------
public class getCDInfo
{
 VideoFormat formats;
  Vector device_list;
  CaptureDeviceInfo CDinfo,
  MediaLocator locator;
  DataSource CCTVSource;
  Processor videoProcessor1;
  Processor videoProcessor2;
  StateConfigurer SCP;
  FileTypeDescriptor FTD;
  DataSink dataSink;
  JInternalFrame jif;

  JButton recordMovie,
  JButton pictureCapture;
  JButton cancel;
  JPanel panel;
  MediaLocator dest;
  DataSource source;
  JComboBox format;
  JComboBox devices;
  private double scaleFactor;
  private BufferToImage bufferToImage;
  BufferedImage image;
  BufferedImage im;
  Buffer buf;
  String deviceName;

  public getCDInfo(final JDesktopPane jdpane, final JFrame frame)
  {
   formats = new VideoFormat(VideoFormat.YUV);

   device_list = CaptureDeviceManager.getDeviceList(formats);

   if(device_list.size() > 0)
   {
      final JDialog dial = new JDialog(frame, "Select Capture Device", true);
    JPanel DPanel = new JPanel();
      JPanel bt = new JPanel();

        dial.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
      dial.setSize(300, 170);
      dial.setModal(true);
        DPanel.setLayout(null);
        DPanel.setBorder(new TitledBorder("Capture Device"));
        DPanel.setBackground(Color.white);
      bt.setLayout(new FlowLayout());

      Dimension win = dial.getSize();
      Dimension opr = Toolkit.getDefaultToolkit().getScreenSize();
```

```
int width = (dpr.width-win.width)/2;
int height = (dpr.height-win.height)/2;
dial.setLocation(width, height);

JLabel captureDevs = new JLabel("Capture Devices");
  devices = new JComboBox(device_list);
  devices.setMaximumRowCount(20);
JButton okay = new JButton("Okay");
JButton cancel = new JButton("Cancel");

bt.add(okay);
  bt.add(cancel);

   captureDevs.setBounds(5, 15, 100, 30);

   DPanel.add(captureDevs);

   devices.setBounds(100, 17, 184, 22);

   DPanel.add(devices);
  dial.getContentPane().add(DPanel, BorderLayout.CENTER);
  dial.getContentPane().add(bt, BorderLayout.SOUTH);

  okay.addActionListener(new ActionListener()
  {
    public void actionPerformed(ActionEvent ok)
    {
      CDInfo = (CaptureDeviceInfo)devices.getSelectedItem();
      dial.dispose();
          jif = new JInternalFrame("Capture Direct from Camera", false, true, false, true);
       Container ct = jif.getContentPane();
           CCTVVideoCapture CVC = new CCTVVideoCapture(jif, frame, jdpane, CDInfo);

        ct.add(CVC, BorderLayout.CENTER);

        jif.setSize(600, 500);
        jif.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);
        jif.setBackground(Color.gray);
        Icon ic = new ImageIcon("LOGO1.gif");
        jif.setFrameIcon(ic);

        Dimension win = jif.getSize();
        Dimension dpr = Toolkit.getDefaultToolkit().getScreenSize();
        int width = (dpr.width-win.width)/2;
        int height = (dpr.height-win.height)/6;

        jif.setLocation(width, height);
        try
        {
          jif.setSelected(true);
        }
        catch(Exception ecf){ecf.toString();}
        jdpane.setSelectedFrame(jif);
        jdpane.add(jif);
        jif.setVisible(true);
      }
     });

     cancel.addActionListener(new ActionListener()
     {
```

```
            public void actionPerformed(ActionEvent can)
            {
                dial.dispose();
            }
            });

            dial.setVisible(true);
        }
        else
        {
                JOptionPane.showMessageDialog(null, "Your camera is not connected", "No video camera found",
                JOptionPane.WARNING_MESSAGE);
            return;
        }
    }
}
```

- **CCTVVideoCapture.java**

```
//importing/including external libraries and code files
//------------------------------------------
import javax.swing.*;
import javax.swing.JOptionPane *;
import javax.swing.border.*;
import javax.swing.BorderFactory.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.awt.Component;
import javax.swing.JOptionPane;
import javax.media.*;
import javax.media.protocol.*;
import javax.media.format.VideoFormat;
import javax.media.format.AudioFormat;
import java.awt.event.*;
import javax.swing.event.*;
import java.net.MalformedURLException;
import javax.swing.JFileChooser;
import javax.media.control.*;
import javax.media.Manager.*;
import javax.media.format.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.media.util.BufferToImage.*;
import javax.media.util.*;
//------------------------------------------------------------

public class CCTVVideoCapture extends JPanel implements Runnable
{
 VideoFormat formats;
 Vector device_list;
 MediaLocator locator;
 DataSource CCTVSource;
 Processor videoProcessor1;
 Processor videoProcessor2;
 StateConfigurer SCF;
 FileTypeDescriptor FTD;
```

```
DataSink dataSink;
CaptureDeviceInfo cdi;

JButton recordMovie;
JButton pictureCapture;
JButton cancel;
JPanel panel;
MediaLocator dest;
DataSource source;
JComboBox format;
JComboBox devices;
private double scaleFactor;
private BufferToImage bufferToImage;
BufferedImage image;
BufferedImage im;
Buffer buf;
String deviceName;
String FName;
String DIR;

Thread logot;

public void run()
{
  while(true)
  {
    this.repaint();
    try
    {
      Thread.sleep(100);
    }
    catch(Exception e){}
  }
}

public CCTVVideoCapture(final JInternalFrame own, final JFrame owner, final JDesktopPane jdpane, final
CaptureDeviceInfo CDInfo)
{
  logot = new Thread(this);
  this.setLayout(new BorderLayout());
  panel = new JPanel();
  panel.setLayout(new FlowLayout());
  cdi = CDInfo;

  try
  {
      locator = CDInfo.getLocator();
     CCTVSource = Manager.createDataSource(locator);
     CCTVSource = Manager.createCloneableDataSource(CCTVSource);
  }
  catch(IOException e)
  {
        JOptionPane.showMessageDialog(null, "The Capture Device you selected might currently be in
        use.\nPlease select another Capture device or Close the process making use\nof this capture device",
        "Capture Device in Use", JOptionPane.WARNING_MESSAGE);
     own.setVisible(false);
     return;
  }
  catch(Exception e)
  {
```

```
        JOptionPane.showMessageDialog(null, "The Capture Device you selected might currently be in
        use.\nPlease select another Capture device or Close the process making use\nof this capture device",
        "Capture Device in Use", JOptionPane.WARNING_MESSAGE);
      own.setVisible(false);
      return;
    }

    try
    {
      videoProcessor1 = Manager.createProcessor(CCTVSource);
    }
    catch (IOException e)
    {
          JOptionPane.showMessageDialog(null, "IO Exception creating processor: " + e.getMessage(), "Error",
          JOptionPane.WARNING_MESSAGE);
      return;
    }
    catch (NoProcessorException e)
    {
          JOptionPane.showMessageDialog(null, "Exception creating processor: " + e.getMessage(), "Error",
          JOptionPane.WARNING_MESSAGE);
      return;
    }

    SCF = new StateConfigurer(videoProcessor1);
    if(!SCF.configure(10000))
    {
          JOptionPane.showMessageDialog(null, "cannot configure processor", "Error",
          JOptionPane.WARNING_MESSAGE);
      return;
    }

      videoProcessor1.setContentDescriptor(null);

      if(!SCF.realize(10000))
      {
            JOptionPane.showMessageDialog(null, "cannot realize processor", "Error",
            JOptionPane.WARNING_MESSAGE);
        return;
      }
      videoProcessor1.start();
      videoProcessor1.getVisualComponent().setBackground(Color.gray);

          pictureCapture = new JButton("Take Snapshot");
      pictureCapture.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent PCT)
        {
          if (videoProcessor1 == null)
          {
            return;
          }
          videoProcessor1.stop();
          videoProcessor1.deallocate();
          videoProcessor1.close();

          final JDialog dial = new JDialog(owner, "Take Snapshot", true);
          JPanel DPanel = new JPanel();
          JPanel bt = new JPanel();
          dial.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
```

```
dial.setSize(300, 170);
dial.setModal(true);
DPanel.setLayout(null);
DPanel.setBorder(new TitledBorder("Save Picture"));
DPanel.setBackground(Color.white);
bt.setLayout(new FlowLayout());

Dimension win1 = dial.getSize();
Dimension opr1 = Toolkit.getDefaultToolkit().getScreenSize();
int width1 = (opr1.width-win1.width)/2;
int height1 = (opr1.height-win1.height)/2;
dial.setLocation(width1, height1);

JLabel captureDevs = new JLabel("Enter File Name:");
final JTextField filename = new JTextField(10);
JButton okay = new JButton("Okay");
JButton cancel = new JButton("Cancel");

bt.add(okay);
bt.add(cancel);
captureDevs.setBounds(5, 15, 100, 30);
DPanel.add(captureDevs);
filename.setBounds(160, 17, 184, 22);
DPanel.add(filename);
dial.getContentPane().add(DPanel, BorderLayout.CENTER);
dial.getContentPane().add(bt, BorderLayout.SOUTH);

okay.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent ok)
    {
      try
      {
                ObjectInputStream istream = new ObjectInputStream(new
                BufferedInputStream(new FileInputStream("picDIR.ocf")));
        DIR = (String)istream.readObject();
        FName = DIR+filename.getText();
      }
      catch(Exception stre)
      {
                JOptionPane.showMessageDialog(null, "Error reading URL", "Error",
                JOptionPane.ERROR_MESSAGE);
        stre.printStackTrace();
      }
      dial.dispose();

      JMFCapture camera;
      camera = new JMFCapture(300, FName);

      camera.close(); // close down the camera
      own.dispose();
  }
});
cancel.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent can)
    {
      dial.dispose();
```

```
                own.dispose();
              }
            });
            dial.setVisible(true);
               JInternalFrame jif = new JInternalFrame("Capture Direct from Camera", true, true, true, true);
            Container ct = jif.getContentPane();
                 CCTVVideoCapture1 CVC = new CCTVVideoCapture1(jif, owner, jdpane, CDInfo);
             ct.add(CVC, BorderLayout.CENTER);
             jif.setSize(600, 500);
             jif.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);
             jif.setBackground(Color.gray);
             Icon ic = new ImageIcon("LOGO1.gif");
             jif.setFrameIcon(ic);

             Dimension win = jif.getSize();
             Dimension opr = Toolkit.getDefaultToolkit().getScreenSize();
             int width = (opr.width-win.width)/2;
             int height = (opr.height-win.height)/6;
             jif.setLocation(width, height);
             jdpane.setSelectedFrame(jif);
             jdpane.add(jif);
             jif.setVisible(true);
          }
        });

        recordMovie = new JButton("Record Movie");
        recordMovie.addActionListener(new ActionListener()
        {
          public void actionPerformed(ActionEvent EET)
          {
            if (videoProcessor1 == null)
            return;
            videoProcessor1.stop();
            videoProcessor1.deallocate();
           ' videoProcessor1.close();
            VideoCapture capture = new VideoCapture(owner, own, jdpane, CDInfo);
          }
        });
        cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener()
        {
          public void actionPerformed(ActionEvent canc)
          {
            if (videoProcessor1 == null)
            return;
            videoProcessor1.stop( );
            videoProcessor1.deallocate( );
            videoProcessor1.close();
            own.dispose();
          }
        });
        panel.add(recordMovie);
        panel.add(pictureCapture);
          panel.add(cancel);
          this.add(videoProcessor1.getControlPanelComponent(), BorderLayout.NORTH);
          this.add(videoProcessor1.getVisualComponent(), BorderLayout.CENTER);
          this.add(panel, BorderLayout.SOUTH);

          logot.start();
    }
```

```
private void saveSnap(BufferedImage im, String fnm)
// Save image as JPG
{
  System.out.println("Saving image");
  try
  {
    ImageIO.write(im, "jpg", new File(fnm+".jpg"));
      JOptionPane.showMessageDialog(null, "Image Saved Successfully", "Success",
      JOptionPane.INFORMATION_MESSAGE);
  }
  catch(IOException e)
  {
        JOptionPane.showMessageDialog(null, "Could not save Image", "Not Successful",
        JOptionPane.INFORMATION_MESSAGE);
      e.printStackTrace();
  }
}

class JMFCapture
{
  private int size = 300;
  private Player p;
  private FrameGrabbingControl fg;
  private boolean closedDevice;
  private static final String CAP_LOCATOR = "vfw://0";
  CaptureDeviceInfo devInfo = null;

  private MediaLocator findMedia()
  {
    Vector devices = CaptureDeviceManager.getDeviceList(null);
    devInfo = (di);  //(CaptureDeviceInfo) deviceName;
    return devInfo.getLocator(); // this method may not work
  } // end of findMedia()

  public JMFCapture(int sz, String FName)
  {
    size = sz;
    closedDevice = true; // since device is not available yet
    // link player to capture device
    try
    {
      MediaLocator ml = findMedia();
      p = Manager.createRealizedPlayer(ml); // stage 2
      System.out.println("Created player");
    }
    catch (Exception e)
    {
      System.out.println("Failed to create player");
      return;
    }
    // create the frame grabber (stage 3)
      fg = (FrameGrabbingControl)        p.getControl("javax.media.control.FrameGrabbingControl");
    if (fg == null)
    {
      System.out.println("Frame grabber could not be created");
      return;
    }
    System.out.println("Frame Grabber created");
    // wait until the player has started (stage 4)
```

```
        System.out.println("Starting the player...");
        try
        {
          p.start();
          System.out.println("player Started");
        }
        catch (Exception e2)
        {
          e2.printStackTrace();
        }

        waitForBufferToImage(); // stage 5
        System.out.println("BufferToImage: "+bufferToImage);
        System.out.println("Buffer: "+buf);

        Image im = bufferToImage.createImage(buf);
        if (im == null)
        {
          System.out.println("No grabbed image");
        }
        else
        {
          System.out.println("Image: "+im);
        }
        image = makeBIM(im);
        saveSnap(image, FName);
    } // end of JMFCapture()

    void waitForBufferToImage()
    {
      boolean result = hasBufferToImage();

      while(!result)
      {
        result = hasBufferToImage();
      }
    }

    private boolean hasBufferToImage()
    {
      buf = fg.grabFrame(); // take a snap
      while (buf == null)
      {
        System.out.println("No grabbed frame");
        return false;
      }
      // there is a buffer, but check if it's empty or not

      VideoFormat vf = (VideoFormat) buf.getFormat();
      while(vf == null)
      {
        System.out.println("No video format");
        return false;
      }
      System.out.println("Video format: " + vf);

      int width = vf.getSize().width;
      int height = vf.getSize().height;
      if (width > height)
      scaleFactor = ((double) size) / width;
```

```
        else
        scaleFactor = ((double) size) / height;
        // initialize bufferToImage with the video format info.
        bufferToImage = new BufferToImage(vf);
        return true;
      } // end of hasBufferToImage()

      private BufferedImage makeBIM(Image im)
      {
        BufferedImage copy = new BufferedImage(size, size, BufferedImage.TYPE_INT_RGB);
        // create a graphics context
        Graphics2D g2d = copy.createGraphics();
        // image --> resized BufferedImage
        setScale(g2d, im);
        g2d.drawImage(im,0,0,null);
        g2d.dispose();
        return copy;
      } // end of makeBIM()

      private void setScale(Graphics2D g2d, Image im)
      {
        if (scaleFactor == 0.0) { // scale not yet set
        int width = im.getWidth(null); // get the image's dimensions
        int height = im.getHeight(null);
        if (width > height)
        scaleFactor = ((double) size) / width;
        else
        scaleFactor = ((double) size) / height;
        }
        g2d.scale(scaleFactor, scaleFactor); // scale the context
      }

      synchronized public BufferedImage grabImage()
      {
        if (closedDevice)
        return null;
        System.out.println("BufferToImage: "+bufferToImage);
        Image im = bufferToImage.createImage(buf);
        System.out.println("Image: "+im);
        if (im == null)
        {
          System.out.println("No grabbed image");
          return null;
        }
        return makeBIM(im), // image --> BufferedImage
      } // end of grabImage()

      synchronized public void close()
      {
        p.close();
        closedDevice = true;
      }
    }
  }
```

Due to maximum page constraints, the codes where not listed in their entirety.

CHAPTER FOUR

Tests, Results and Discussion

4.1 Viewing and Capturing Video and Audio from Capture Devices

To view from the Video Camera, launch the software (Digital Eye) then click on the "Action" menu and select the "View video from Cam" menu item. You are prompted with a dialog box containing a combo box with a list of the Video cameras and other capture devices like Microphones connected to your computer, selecting any of the capture devices and clicking on the "Okay" button disposes the dialog box after which you are let to view or listen to the output from the capture device. If the capture device is a Video Camera, you will see a live video being displayed on the opened window else if it was a microphone you will hear your voice from the speakers of the computer.

You can capture the current event and save it as a file on your system for playback later. Video files are captured and saved if you are using a Video Camera. You can also take snapshots with the Video Camera and save it to your system as a Picture file (JPEG).

To capture Video from the camera, click on the "Record Movie" in the open window (i.e. the View video from Cam menu item). On clicking this button, you are prompted with a dialog box packed with various settings needed to compose or build up a video. A screen shot of the dialog box titled "Capture Video" is presented below:

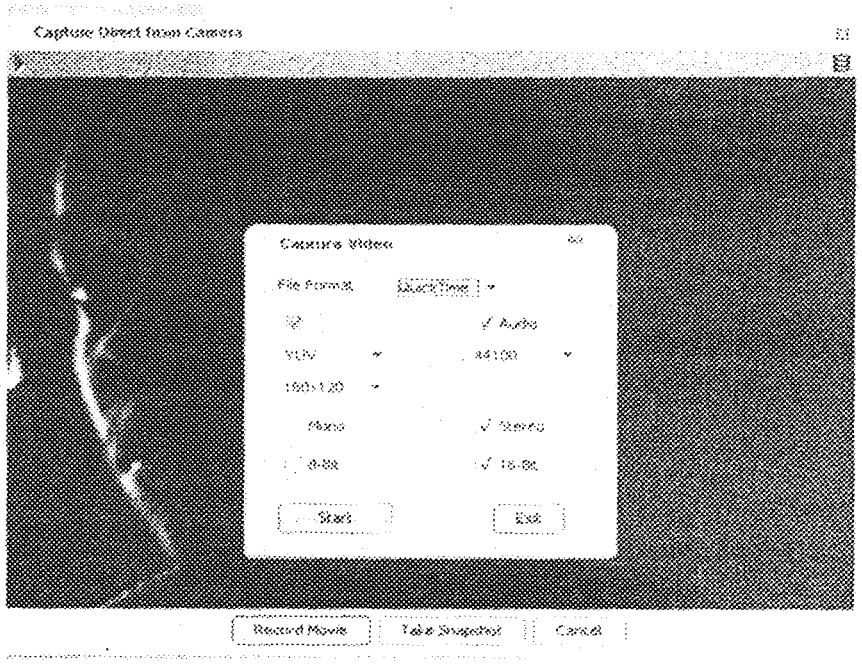


Fig.4.1 Capture video dialog box

The settings present in the dialog box are:

* **File Format:** This setting deal with the extension of the video file which determines the application that can playback the video file. E.g. the Quicktime format (.MOV) is originally designed to playback with the Quicktime Application from Apple Computers, so if you don't have this application installed, you might not be able to play back a video file with this format. Some other applications playback this format like the Nero Media Player. Other formats supported by this Project (Software) include the Video for Windows format (.AVI) and MPEG format (.mpg)

* **Video Settings:** This setting deals with the format which the Video is encoded with and the screen resolution (screen area which the video occupies) of the video.

The Video formats available in this software includes: YUV, RGB, CINEPAK, H.261, H.263 and the resolutions available are: 160 x 120, 320 x 240, 640 x 480, 800 x 600, and 1024 x 768. Note that not all formats and resolution listed above might be supported by your camera. Test each of the format and resolution to get your desired resolution which is supported by your camera.

* **Audio Settings:** The audio settings contain information needed for encoding the audio within the video, information's like the frequency which the audio plays, the bit rate and the sound quality (Mono or Stereo). If after capturing you don't hear any audio while the video plays back then your camera might not have a microphone or sound system built into it. You can attach an external microphone to your system to serve as your sound capture facility.

Satisfied with the settings, you can then go on to click on the start button to begin capturing. Clicking this button presents a dialog box asking you to enter the name which you wish to give the video file. After entering the name, the capture begins.

When you are done with capturing, click on the End button to stop the capture process. The "Start" and "Exit" buttons will be enabled once again along with other components on the dialog box. To exit the "Capture Video" dialog box, click on the exit button. This button closes the dialog and re-opens the live video view from the Video camera.

4.2 Taking Snapshots

To take a snapshot of the Camera's immediate view, click on the "Take Snapshot" button. On clicking this button, the picture is taken and you are prompted to enter the filename for the picture file. Note that depending on the speed of your camera and Video formats supported by your camera, there might be a slight delay in the picture capture. If your camera is a bit slow and supports few video formats, the program takes some time to locate the video format supported by the camera before it snaps. If the Snapshot process was successful, you will get a message telling you that the image was successfully saved else you get a message telling that there was an error during the process.

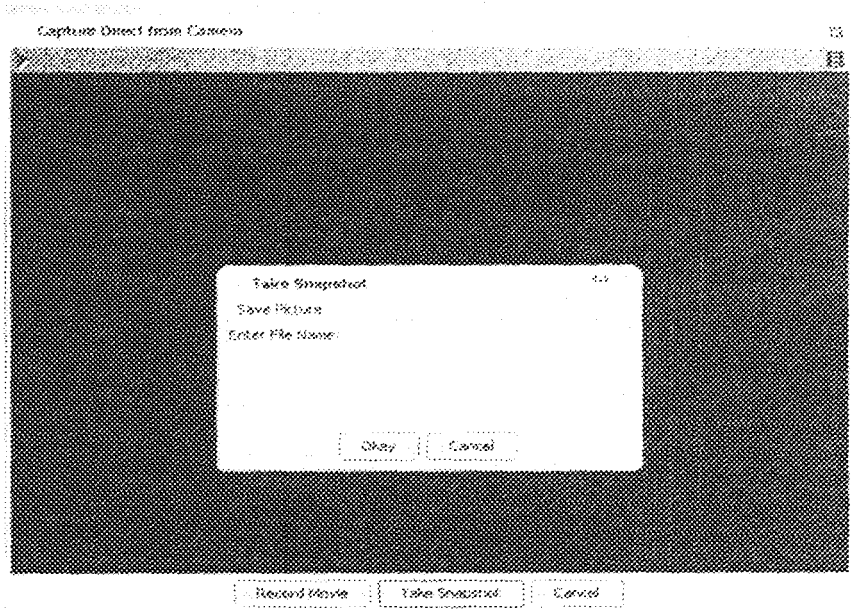


Fig 4.2 Taking snapshots

Clicking on the pause button at the upper left part of the window pauses the live view from the camera. To exit this window, click on the "Cancel" button

4.3 Playback of Video and Audio files

For this functionality, click on the "Actions" menu and select the "Play Captured Videos" menu item. After selecting this menu item you are prompted with an "Open" dialog box to enable you select the Video or Audio file which you want to playback. Note that this project dissertation does not support all video or audio files. Video files supported include files with extension .MOV, .AVI, .mpg, .SWF and a host of others but the popular .DAT format for VCD (Video Compact Disk) files is not supported. Audio files supported include .mp3, Sun Audio (.au) and a host of others but formats like The Windows Media Audio (.WMA) is not supported. All video captured with this software is supported.

After selecting the video file of your choice, click on the "Open" button within the dialog. This begins the playback of the video. A window opens up with the video being played back. The video controls are located at the top of the window. To stop the playing video, click on the "Stop" button at the lower end of the window and to close this window, click on the "Cancel" button. Below is a screen shot of a playing video.

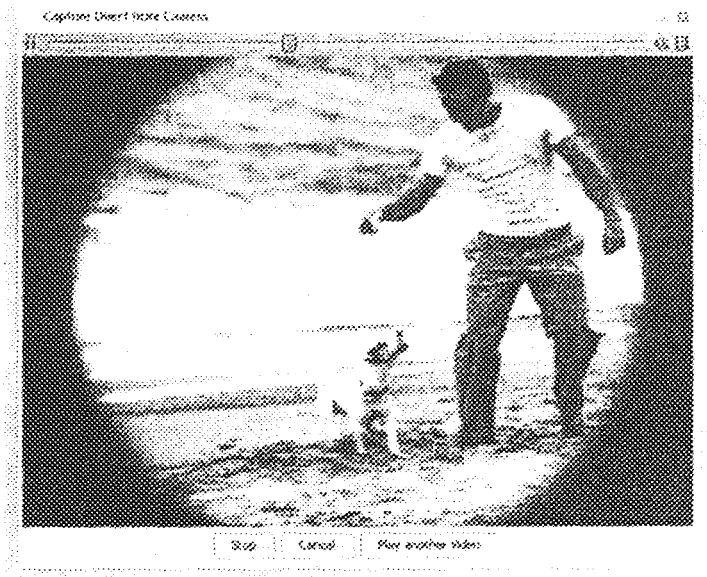


Fig.4.3 playing video

To play another video or change the current playing video, click on "Play another Video" button then select another video.

4.4 Setting the output directories for Video and picture files captured

You can set the location where your video and picture files are stored each time you capture and save videos and audios.

To set the output directories for video and picture files captured, select the "Options" menu item from the "Actions" menu. A dialog box is opened with two text fields on it, one holding the path where your captured pictures are and the other holding the path where your captured videos are. To change this, click on the "Browse" button beside each text field and select the directory (folder) where you want your files to be stored then click on the "Okay" button to save your changes. The dialog box looks like this:

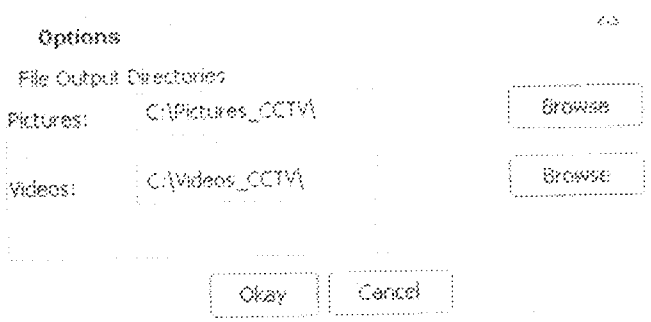


Fig.4.4 Options dialog box

CHAPTER FIVE

Conclusions and Recommendations

5.1 Conclusions.

The objective of this project which was realized was to create a program that will capture and playback video received from a video camera. The source code was generated and compiled and virtually all aspects went according to plan. The code bridges the system and the external peripheral which in this case is a video camera required to feed data into the system via the universal serial bus (USB) to a system operating on the Windows platform.

The program was written in java which is a programming language created and updated by Sun Microsystems.

5.2 Possible improvements.

Further work could be done in the following area to enhance the softwares functionality

1. Capture could be configured to automatically capture snapshot images, or record video and audio images. A hotkey could be could be incorporated to take handle such. One press from a designated hotkey initiates image capture and storage.

2. A login process could be designed to resrict access to the softwares capabilities; the flow charts shown in appendix 1and 2could very well do the job.

3. The program could be enhanced to detect motion in monitored areas; it sounds an alarm, stores captured images or records video when motion is detected.

References

[1] http://en.wikipedia.org/wiki/universal serial bus

[2]Article by Pramod Jain and Yayati Kasralikar, Web conferencing using JMF

[3]Article by Gal Ratner, Focus on Java, inverted Software page1

[4]D&A Deitel and Associate, Java how to program, , published by Prentice Hall pg1246

[5] Sun Microsystems, java Media Frame Work API Guide, Nov 19 1999

Miscellaneous references

Article by David Fischer, Java programming examples, 23rd Jan 2003

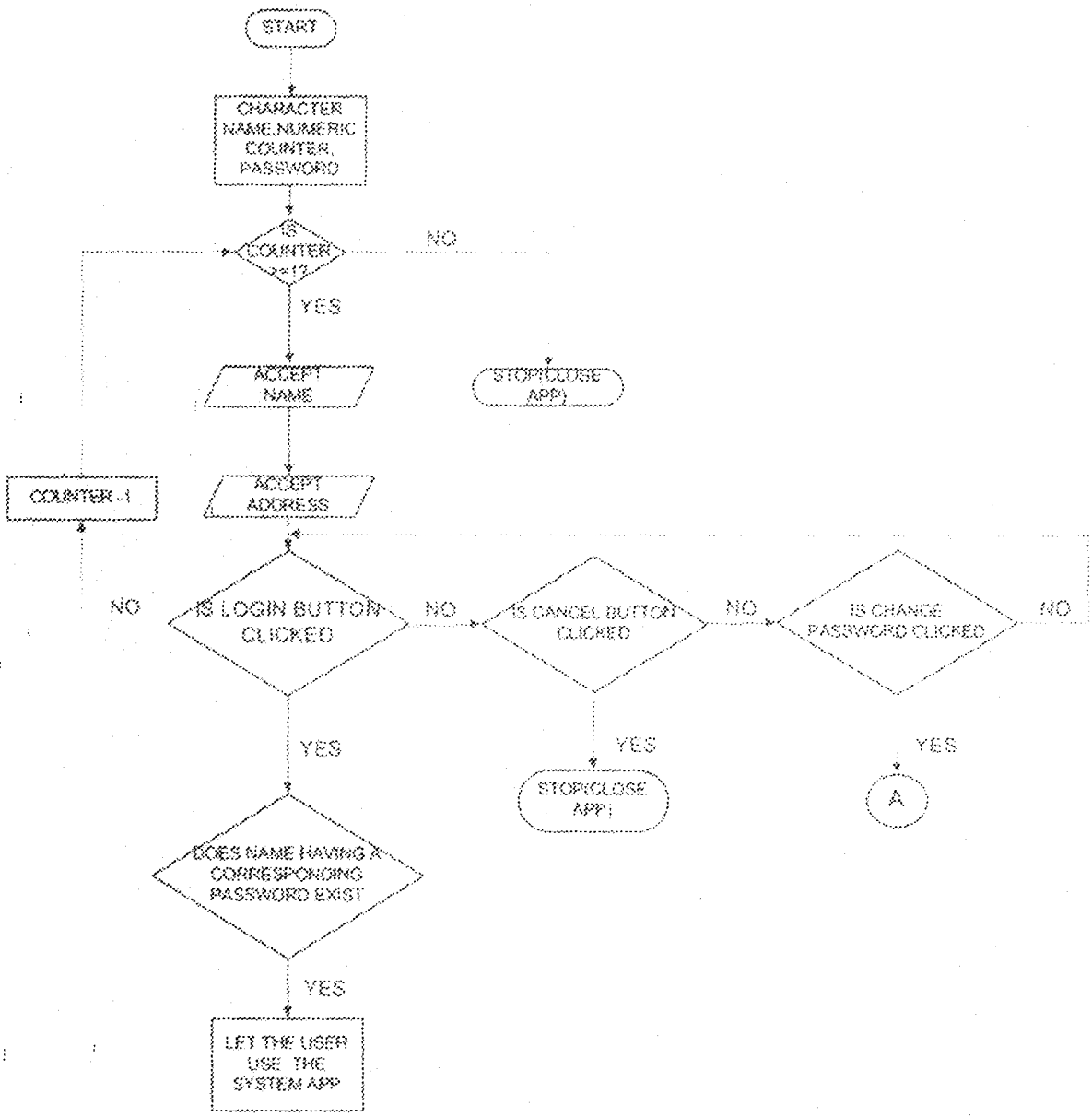
Article by Budi Kurniawan, posted on www.javaworld.com/feedback

Elliotte Rusty Harold, Java I/O, First Edition March 1999

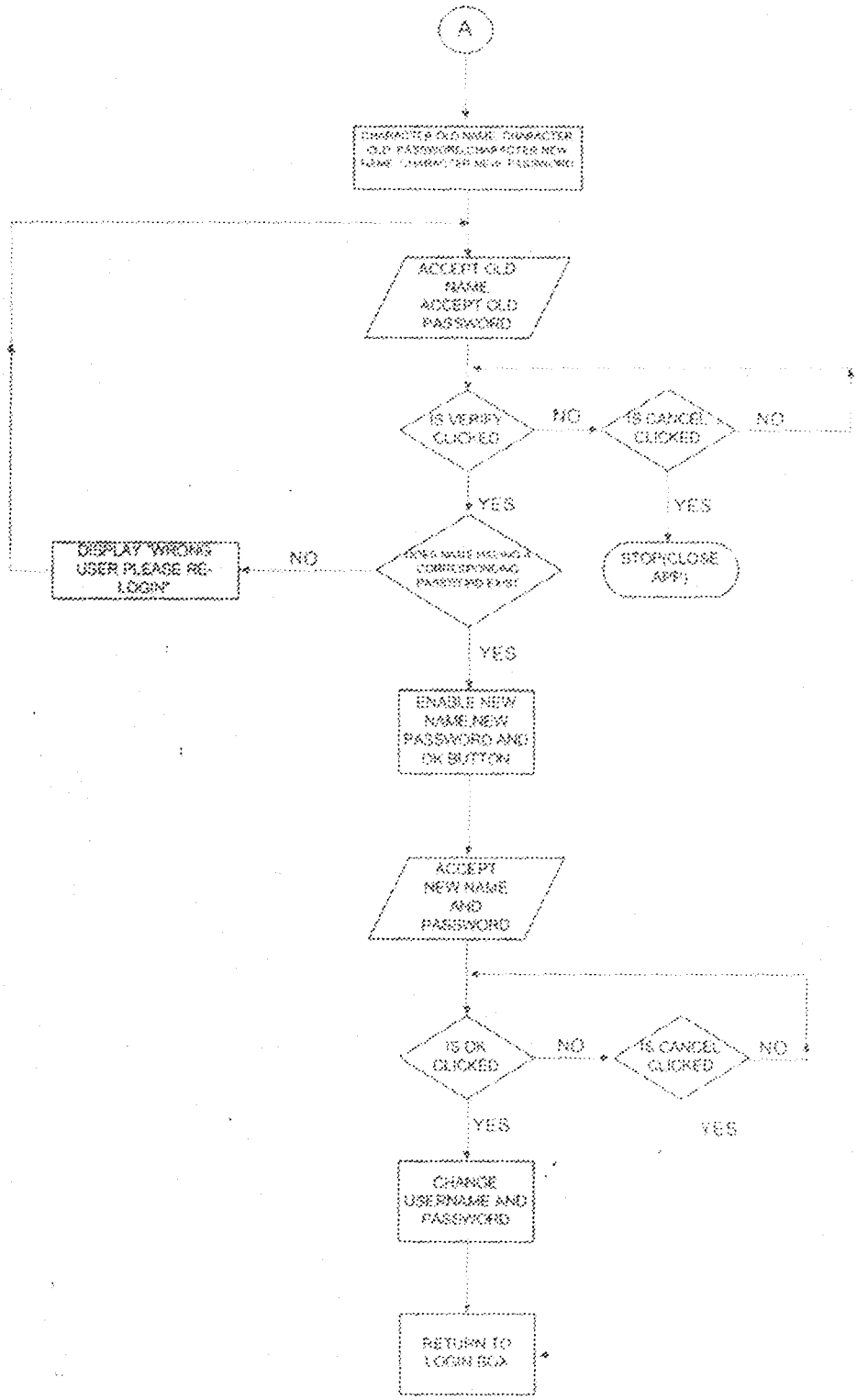
Bruce Eckel, Thinking in Java second edition, published by Prentice-Hall mid-June, 2000

Sun Microsystems, the Java tutorial, published by Sun Microsystems.

Flowcharts



Appendix 1


A
ACCEPT OLD NAME
ACCEPT OLD PASSWORD
IS VERIFY CLICKED
NO
IS CANCEL CLICKED
NO
YES
YES
DISPLAY "WRONG USER PLEASE RE-LOGIN"
NO
STOP(CLOSE APP)
YES
ENABLE NEW NAME,NEW PASSWORD AND OK BUTTON
ACCEPT NEW NAME AND PASSWORD
IS OK CLICKED
NO
IS CANCEL CLICKED
NO
YES
YES
CHANGE USERNAME AND PASSWORD
RETURN TO LOGIN BOX

Appendix 2