



**Jaypee University of Information Technology**  
**Solan (H.P.)**  
**LEARNING RESOURCE CENTER**

Acc. Num. *SP06075* Call Num:

**General Guidelines:**

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT



**SP06075**

**INFRARED BASED VIDEO SURVEILLANCE  
ON MOBILE PHONES**

By

**AYUSH AGARWAL - 061225**

**BHUNESHWAR KUMAR – 061313**

**ANCHIT BANSAL – 061325**

**SIDDHANT UPPAL – 061331**



**DEPARTMENT OF COMPUTER SCIENCE  
AND ENGINEERING**

**JAYPEE UNIVERSITY OF INFORMATION  
TECHNOLOGY – WAKNAGHAT**

**MAY – 2010**

## CERTIFICATE

This is to certify that the work entitled "**Infrared based Video Surveillance on Mobile Phones**" submitted by Ayush Agarwal (061225) Bhuneshwar Kumar (061331) Anchit Bansal (061325) and Siddhant Uppal (061331) for the award of Bachelor of Technology in Computer Science and Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.



**Brig (Retd.) S.P. Ghreera**

Head of Department

Computer Science Engineering and Information Technology

Jaypee University of Information Technology


Waknaghat.


## ACKNOWLEDGEMENTS


We would like to extend our gratitude and take this opportunity to thank our esteemed project guide, **Brig (Retd.) S.P. Ghrera** who helped us time and again and also guided us in times of uncertainty.

We would also like to thank **Mr. Satish Chandra** (CSE) who helped us and gave us new innovative ideas for extending the scope of our project and increasing the security provided by the surveillance system.

We would also like to thank **Mr. Pramod** and **Mr. Pandey**, incharge of the Microprocessor Laboratory and the Electronics Laboratory respectively for their help in implementing the circuit.

  
Ayush Agarwal  
061225

  
Bhuneshwar Kumar  
061313

  
Ankit Bansal  
061325

  
Siddhant Uppal  
061331

## TABLE OF CONTENTS

- A. List of Figures
- B. List of Abbreviations
- C. Abstract
  
- 1. Problem Statement
  - 1.1. Description
    - 1.1.1. Setting up the Infrared Infrastructure
    - 1.1.2. Connectivity with the speaker
    - 1.1.3. Connectivity with the Web camera
    - 1.1.4. Video Capturing
    - 1.1.5. Video Saving
    - 1.1.6. Video Transfer
    - 1.1.7. Image Acquisition
    - 1.1.8. Image Saving
    - 1.1.9. Image Transfer
    - 1.1.10. Sending Warning SMS
- 2. Objective and Scope of the Project
- 3. Initial Project Investigation
  - 3.1. Home security system for detecting an intrusion into a monitored area by an infrared detector by Devan Dockery
  - 3.2. Infrared alarm system by Milton O. Smith
  - 3.3. Outdoor infrared video surveillance: A novel dynamic technique for the subtraction of a changing background of IR images.
  - 3.4. IR based third eye implementation for industrial safety with loud 60dB siren.
- 4. Motivation
- 5. Hardware Components
  - 5.1. Block Diagram
  - 5.2. Block Description

- 5.2.1. Infrared Transmitter
- 5.2.2. Infrared Receiver
- 5.2.3. Relay
- 5.2.4. Speaker
- 5.2.5. Webcam
- 5.3. Circuit Diagram
- 5.4. Circuit Description
- 5.5. List of Components
- 6. Software Components
  - 6.1. Overview
  - 6.2. Sub Process Description
    - 6.2.1. Image Acquisition
    - 6.2.2. Image Saving
    - 6.2.3. Image Transfer
    - 6.2.4. Sending SMS
- 7. Testing Techniques in Real Time
- 8. Contribution of the Project
- 9. Conclusions
- 10. References
- 11. Appendix
  - 12.1. Main code for detection of the webcam, the video streaming, image capturing, image saving, video saving, SMS sending and backup creation.
  - 12.2. Code for file transferring through bluetooth.
  - 12.3. Sending SMS
  - 12.3. Backend code for various components-buttons, menu, video space, etc.
  - 12.4. Code for password authentication
  - 12.5. Backend code for various buttons, text boxes, etc.

## LIST OF FIGURES

- Fig 1 : Block diagram of the surveillance system
- Fig 2 : IR transmitter circuit's implementation diagram
- Fig 3 : IR receiver circuit's implementation diagram
- Fig 4 : Relay circuit's implementation diagram
- Fig 5 : Full circuit diagram (transmitter and receiver)
- Fig 6 : IC NE555 diagram
- Fig 7 : IC NE555 pin diagram
- Fig 8 : IR LED (light emitting diode) diagram
- Fig 9 : IR receiver sensor TSOP 1738 diagram
- Fig 10 : IC UM66 diagram
- Fig 11 : Twisted pair for USB cabling diagram

## LIST OF ABBREVIATIONS

- i. IRVSMS : InfraRed based Video Surveillance on Mobile phones System
- ii. DFD : Data Flow Diagram
- iii. DFG : Data Flow Graph
- iv. ID : identity
- v. LCD : Liquid Crystal Display
- vi. IC : Integrated Circuit
- vii. LAN : Local Area Network
- viii. CPU : Central Processing Unit
- ix. IP : Internet Protocol
- x. AC : Alternating Current
- xi. DC : Direct Current
- xii. SMS : Short Messaging Service
- xiii. AP : Access Point
- xiv. IR : Infrared
- xv. I/O : Input / Output
- xvi. TTL : Transistor Logic
- xvii. RAM : Random Access Memory
- xviii. ROM : Read Only Memory
- xix. GSM : Global System for Mobile communications
- xx. ESP : Enterprise Service Provider
- xxi. LED : Light Emitting Diode
- xxii. USB : Universal Serial Bus
- xxiii. CMOS: Complementary Metal Oxide Silicon.



## **Abstract**

This document will give you an insight in the implementation of “Infrared based Video Surveillance on Mobile Phones.”

In this document we have described the basic steps involved in providing security by making an infrared based surveillance system and then transmitting the live feed of the intrusion back to the owner of the surveillance system.

The project literature forms an important part of this document as we have explained in detail the work performed in this area.

Followed by the description of the project we have given an overview of the concepts used in the implementation of the project.

Further, we have developed a code for the security surveillance and detecting the intrusion and informing about the same to the owner of the surveillance system.

Then we have tried implementing the project in real time and we have explained the problems and limitations of the product being produced and tested our product in real time.

Finally, we draw a conclusion and show our result.

This project takes the advantage by studying the flaws of similar products and works in this area of technology and tries to overcome them in real time.

# CHAPTER 1

## PROBLEM STATEMENT

Security is always an issue at your workplace and house. There are a number of expensive security systems available in the market but in a one way or the other they lack the software component.

We have made a surveillance security system which integrates both hardware and software thus giving us all the processing power of a computer to perform various functions.

### 1.1. DESCRIPTION

The surveillance system is designed to detect any intrusion in a restricted region, then capturing the image of the intruder and informing about the same to the owner.

The basic steps involved in performing this are:

#### 1.1.1. SETTING UP THE INFRARED INFRASTRUCTURE

The infrared infrastructure involves the creation of a transmitter circuit that transmits the IR rays and the creation of a receiver circuit that will receive the IR rays sent by the transmitter circuit. The software built will monitor both these circuits and will provide additional features like image capturing, video capturing, image saving, video saving, SMS sending, etc.

#### 1.1.2. CONNECTIVITY WITH THE SPEAKER

A speaker (alarm) is interfaced with the receiver circuit in such a way that when the IR beam sent by the transmitter circuit is cut or intercepted, the speaker (alarm) will go off automatically and will alert the owner regarding the intrusion.

### **1.1.3. CONNECTIVITY WITH THE WEB CAMERA**

The web camera is installed (interfaced) with the receiver circuit in such a way that when the IR beam sent by the transmitter circuit is cut or intercepted, the web camera starts the video recording of the surveillance region automatically, which is saved in the monitoring computer.

### **1.1.4. IMAGE CAPTURING**

Screenshots of the video that is being streamed is taken automatically once in every three seconds.

### **1.1.5. SENDING WARNING SMS**

After the intrusion has occurred, the owner of the surveillance system is informed about the same by means of sending him an SMS over his mobile phone.

### **1.1.6. IMAGE SAVING**

The screenshots (images) taken above will be saved on the master computer that is monitoring the surveillance system.

### **1.1.7. IMAGE TRANSFERRING**

The image previously captured is sent to the owner's mobile phone. First we connect our mobile phone with the surveillance system via bluetooth and then the image that was previously captured is sent to the owner's mobile phone through our mobile phone by using the services of any GSM service network.

### **1.1.8. VIDEO CAPTURING**

The instant the IR rays are cut that are being transmitted from the IR transmitter circuit to the IR receiver circuit, the web camera is initiated and the video recording i.e. video capturing is started for surveillance purposes.

#### **1.1.9. VIDEO SAVING**

The video that was earlier being captured when the IR beam was intercepted is now being saved on the master computer that is monitoring the surveillance system. This saved video can be now used for further surveillance purposes and repeated playing to identify any intrusion.

#### **1.1.10. VIDEO TRANSFERRING**

The video is transferred from the master computer to the master mobile that has already been synchronized with the master computer via bluetooth. We also have the option of choosing the video which we want to transfer to the master mobile via bluetooth.

## **CHAPTER 2**

### **OBJECTIVE AND SCOPE**

The end product is an infrared based video surveillance system that has excellent security providing and intrusion detecting features like:

1. Image and video capturing,
2. Image and video transferring,
3. Alerting the owner via an SMS.

The end product can be used by Security Service Groups to provide intense security and detect any intrusion in a sensitive region and also capturing the effective details of the intruder.

The small size of the surveillance system increases its portability and also giving it an added advantage that it won't be discovered by any intruder before making his intrusion.

## CHAPTER 3

### INITIAL PROJECT INVESTIGATION

This section gives the details of the previous work done in the field of "Infrared based surveillance systems."

#### 3.1. Home security system for detecting a intrusion into a monitored area by an infrared detector-by Devan Dockery

##### ABSTRACT

A security system has a free-standing intrusion detector. The free standing intrusion detector has a transmitter coupled with a portable receiver to alert a homeowner that an intrusion has taken place or occurred within a pre-set time period. The area under surveillance is monitored by an infrared detector which activates the transmitter upon the detection of abrupt differences in infrared radiation levels, associated with the presence of a warm body in an otherwise equilibrated environment. A radio signal is emitted by the transmitter which is received by the portable hand held remote receiver. A first signal, indicating that an intrusion has been detected less than a preselected period of time in the past in the monitored areas, is displayed on the receiver for that preselected period of time. After the preselected period of time has elapsed, a second signal is generated to indicate that the intrusion took place at a time greater than the preselected period of time in the past and that the probability of the intruder still being present is less. Once the intrusion detector is activated, the signal is continuously transmitted to the portable receiver until the intrusion detection has been reset.

3.2. Infrared alarm system-by Milton O. Smith, Bothell, Washington

ABSTRACT

A battery powered infrared sensor security system capable of operating a single set of batteries for a minimal of one year. The system is connected to a telephone line and employs a bidirectional dual-tone multiple frequency (DTMF) tone generator/receiver to allow communication to and from a remote location. The system status may be checked from a remote location. The system uses a Fresnel lens arrangement and a pair of infrared sensors to provide a substantially uniform field of coverage of 180 degrees. The system also uses real time digital analysis of the output signals from the infrared sensors. The digital analysis uses time sequence analysis of the output signals, perform variance measurements between the current measurement of the infrared sensor signals and the stored time sequence, coherence measurements between the two sensors, and can compare measured amplitude spectra to predefined signature spectra entered by the user.

3.3. Outdoor infrared video surveillance; A novel dynamic technique for the subtraction of a changing background of IR images

ABSTRACT

For security applications, automatic detection and tracking of moving objects is an important and challenging issue especially in uncontrolled environments. Recently, due to the decreasing costs and increasing miniaturization of infrared sensors, the use of infrared imaging technology has become an interesting alternative in such applications.

In this paper, a framework is proposed to detect, track and classify both pedestrians and vehicles in realistic scenarios using a stationary infrared camera. More specifically, a novel dynamic background-subtraction technique to robustly adapt

detection to illumination changes in outdoor scenes is proposed. We noticed that combining results with edge detection enables to reduce considerably false alarms while this reinforces also tracking efficiency. The proposed system was implemented and tested successfully in various environmental conditions.

#### 3.4. IR Based Third Eye Implementation for Industrial Safety with Loud 60dB Siren

##### ABSTRACT

Security is primary concern everywhere and for everyone. Every person wants his home, industry etc to be secured. This project describes a security alarm system that can monitor an industry and home. This is a simple and useful security system and easy to install. There will be an arrangement of IR transmitter and IR receiver to which an LED and buzzer are connected .This IR transmitter and IR receiver are kept at two sides of the door of restricted area. The concept is such that, the IR transmitter transmits a frequency of 38 KHZ and this frequency should be detected or received by IR receiver. If any intruder comes in between this arrangement IR receiver cannot detect the signal and, so led and buzzer connected to IR receiver are activated, which alerts the people surrounding that area.



## CHAPTER 4

### MOTIVATION

We got utmost motivation for making this infrared based surveillance system by the in depth study and analysis of the other surveillance systems available in the market.

Also, the price was a major consideration as the other surveillance systems available in the market are frightfully expensive and unaffordable to the common man. So, we decided to make an affordable surveillance system of our own with the features mention above.

A comparative study of the available surveillance systems in the market along with our own infrared based surveillance system is shown in the table below.

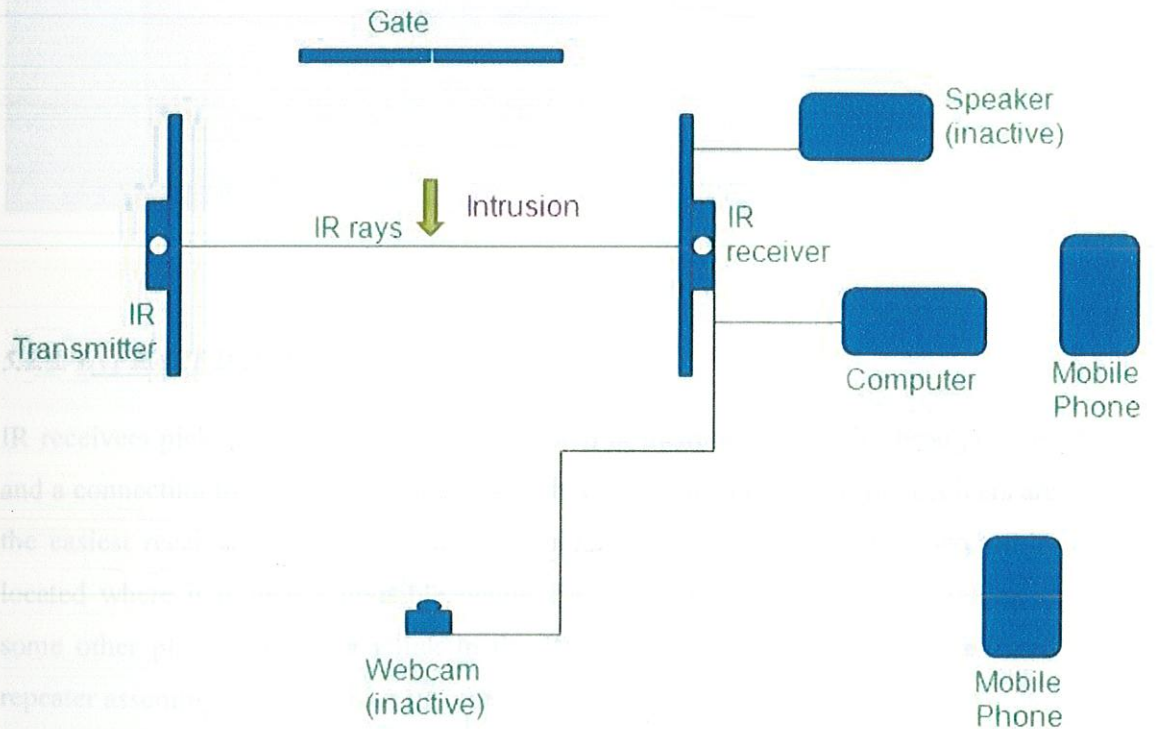
Sr. No.	Name of the System	Price (in INR)	Features Missing
1	Intelligent Wireless Alarm Doorbell # Advante 007-A	3,500/-	No alerts sent on mobile phones. No video streamed.
2	Infrared Security System United States Patent 4751396	2,600/-	No alerts sent on mobile phones. No video streamed.
3	Battery Operated Portable Passive Infrared Intrusion Alarm #AEI PIR-9112	1,800/-	No alerts sent on mobile phones. No video recorded.
4	Electronic Watch Dog	1,670/-	No alerts sent on mobile phones. No video recorded.
5	IRVSMS	800/-	NONE

# CHAPTER 5

## HARDWARE COMPONENTS

### 5.1. BLOCK DIAGRAM

This is the entire pictorial view of our surveillance system's setup.

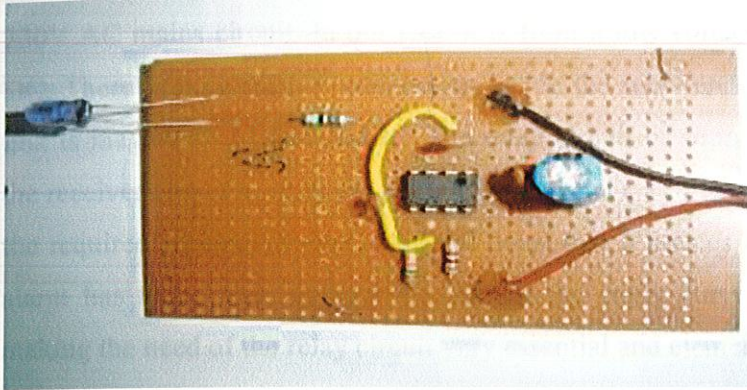


### 5.2. BLOCK DESCRIPTION

#### 5.2.1. INFRARED TRANSMITTER

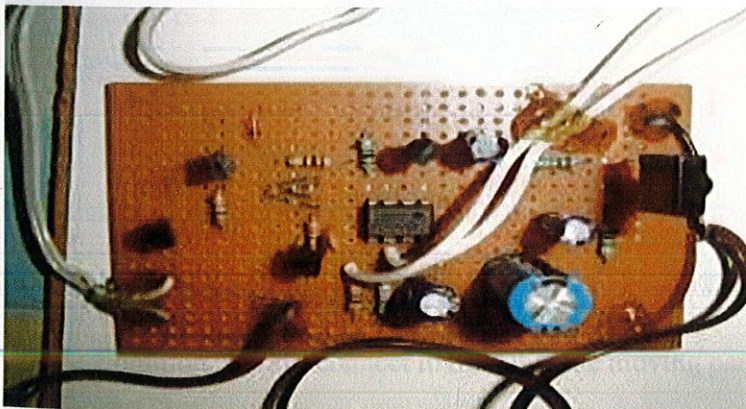
An IR transmitter circuit can be used in many projects as the integral portion of the circuitry hardware. The IR transmitter sends a 40 kHz of frequency carrier under computer control. The computer can turn the IR transmission on and off. The IR carrier at around 40 kHz carries a frequency which is widely used for ICs as

receiving these signals is quite easy. The circuit can be controlled using any TTL (transistor logic) which makes the interfacing very simple. The circuit can be used in many applications, for example, for using a computer to generate IR remote control signals or experimental IR data transmission.



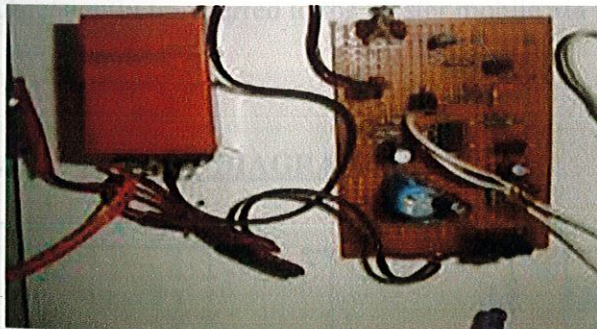
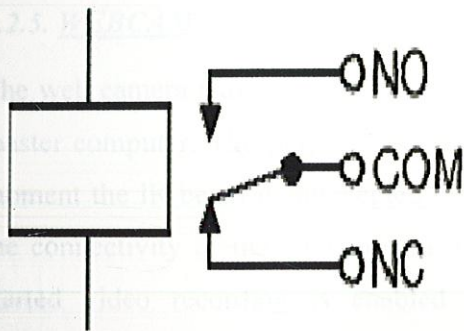
### 5.2.2. INFRARED RECEIVER

IR receivers pickup the infrared beams pointed at them and repeat it, through wires and a connecting block, to infrared emitters that reproduce the signal. IR receivers are the easiest receivers to hide, as their two piece design allows the IR sensor to be located where it is nearly invisible, while the electronics section can be hidden at some other place. The hidden link in the IR receiver is a small shelf-top infrared repeater assembly. It includes an IR receiver and a connecting block.



### 5.2.3. RELAY

The relay is an electro mechanical switch which converts electrical signal into mechanical output and provides the isolation between the two connections. The relay allows one circuit to switch to a second circuit which can be completely separate from the first one. For example, a low voltage DC circuit can use a relay to switch to a 230V AC mains circuit. In our case it is from a low voltage DC to a higher voltage DC. There is no electrical connection inside the relay and between the circuits; the link is magnetic and mechanical. The relay circuit or simply a relay is connected to the receiver circuit to reset the alarm (loudspeaker), as after the alarm has gone off for the required amount of time, the loud noise that it creates is a nuisance. Hence, the alarm has to be reset without turning off the entire surveillance system. Thereby, making the need of the relay circuit very essential and elementary.



In this above relay circuit,

COM : Common, always connect to this; it is the moving part of the switch.

NC : Normally Closed, COM is connected to this when the relay coil is off.

NO : Normally Open, COM is connected to this when the relay coil is on.

### 5.2.4. SPEAKER

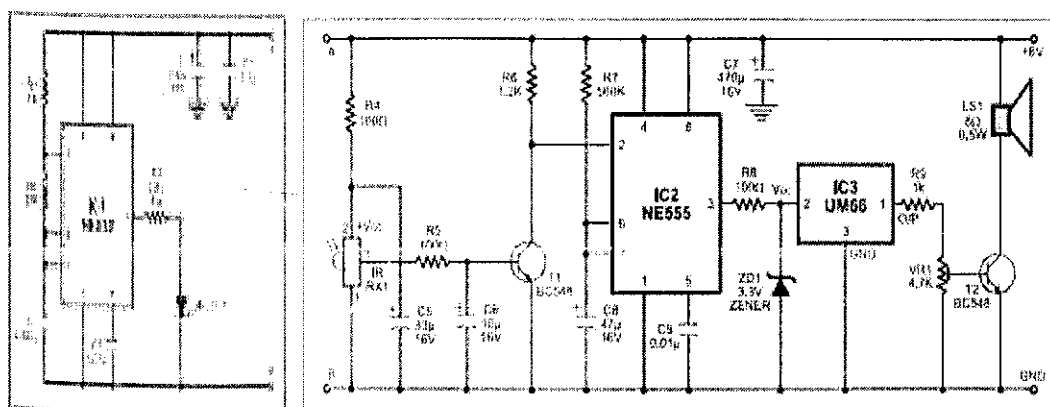
A loudspeaker is an electro-acoustic transducer that converts an electrical signal into sound. The speaker moves in accordance with the variations of an electrical signal and causes sound waves to propagate through a medium such as air or water. After the acoustics of the listening space, loudspeakers are the most variable elements in a modern audio system and are usually responsible for most distortion and audible differences when comparing sound systems.

The loudspeaker is connected to the receiver circuit to function as an alarm and alert the required people near the surveillance system. We were also required to put a volume control to the alarm (loudspeaker) as during the demonstration of the project, the noise that it created was intolerable.

### 5.2.5. WEBCAM

The web camera was required to be connected to the receiver circuit and also to the master computer. The purpose of connectivity with the receiver circuit was that the moment the IR beam is intercepted the video recording is started and the purpose of the connectivity of the web camera with the master computer was that the already started video recording is enabled to be saved on the master computer and subsequently transferred to the master mobile via the bluetooth connectivity.

## 5.3. CIRCUIT DIAGRAM



## 5.4. CIRCUIT DESCRIPTION

### 5.4.1. REGUALTED SUPPLY

A 6 volt battery is used to provide power to the whole branches of the circuit. As the transmitter and the receiver circuit, each work on a 5 volt regulated voltage requirement.

### 5.4.2. TRANSMITTER WORKING

The transmitter circuit is built around timer IC NE555, which is wired as an astable multivibrator producing a frequency of about 38 kHz. The infrared beam is transmitted through IR LED. The power supply for the transmitter is derived from a separate battery by connecting its points A and B to the respective points of the battery. The transmitter and receiver units are aligned such that the IR beam falls directly on the IR sensor.

### 5.4.3. RECEIVER WORKING

The receiver circuit is shown in the figure above. It comprises of an IR sensor TSOP1738, npn transistor BC548, timer IC NE555 and some resistors and capacitors. In the actual configuration, IC2 was wired in a monostable multivibrator with a time period of around 30 seconds but we have increased this time period to 1 min and 8 seconds by changing the value of R7 in the circuit from 560 K ohm to 1.3M ohm, according to the formula:

$$T = RC \ln(3) = 1.1 RC$$

The melody generator section is built around melody generator IC UM66 (IC3), transistor T2 and the loudspeaker. The above figure shows pin configurations of the IR sensor TSOP1738 and melody generator IC UM66. The receiver is powered by regulated 9V DC. For this purpose, we can use a 9V battery. The transmitter and receiver units are aligned such that the IR beam falls directly on the IR sensor. As long as IR beam falls on the sensor, its output remains low, transistor T1 does not conduct and trigger pin 2 of IC2 remains high. When anyone interrupts the IR beam falling on the sensor, its output goes high to drive transistor T1 into conduction and pin 2 of IC2 goes low momentarily. As a result, IC2 gets triggered and its pin 3 goes

high to supply 3.3V to melody generator IC3 at its pin 2, which produces a sweet melody through the speaker, fitted inside the house. Output pin 3 of IC2 remains high for around 68 seconds. We have also installed a RESET switch to switch off the alarm before 68 seconds. The RESET switch is added between pin4 of NE555 IC and ground.

#### **5.4.4. RELAY WORKING**

The relay is used to take more power for the webcam to work properly. When an intrusion is made which triggers off the receiver circuit, we need extra power for webcam to get triggered at the same time and to do so we have put a relay in which we install webcam in the series.

#### **5.4.5. IR RECEIVER SENSOR**

The TSOP 1738 infrared receiver sensor modulates the infrared signal and converts it into a corresponding electrical signal. TSOP 1738 is an infrared receiver sensor which is widely used in a large number of electronic products for receiving and demodulating infrared signals. These received demodulated signals can be easily decoded by a microcontroller. Many effective IR proximity sensors are built around the TSOP 1738 module. The TSOP module is commonly found at the receiving end of an IR remote control system. These modules require the incoming data to be modulated at a particular frequency and would ignore any other IR signals. It is also immune to ambient IR light, so one can easily use these sensors outdoors or under heavily lit conditions. Such modules are available for different carrier frequencies from 32 kHz to 42 kHz. In our project, we will be generating a constant stream of square wave signal using the IC NE555 centered at 38 kHz and would use it to drive the IR led. So whenever this signal bounces off the obstacles, the receiver would detect it and change its output. Since the TSOP 1738 module works in the *active-low configuration*, its output would normally remain high and would go low when it detects the signal (the obstacle).

## 5.5. LIST OF COMPONENTS

### 5.5.1. TIMER IC NE555

The timer IC NE555 is an integrated circuit implementing a variety of timer and multivibrator applications. It has been claimed that the 555 gets its name from the three 5 kΩ resistors used in typical early implementations. Depending on the manufacturer, the standard 555 package includes over 20 transistors, 2 diodes and 15 resistors on a silicon chip.

The timer IC NE555 has three operating modes:

- i. **Monostable Mode:** In this mode, the timer IC NE555 functions as a "one-shot". Applications include timers, missing pulse detection, bounce free switches, touch switches, frequency divider, capacitance measurement, pulse-width modulation (PWM), etc. In the monostable mode, the 555 timer acts as a "one-shot" pulse generator. The pulse begins when the 555 timer receives a signal at the trigger input that falls below a third of the voltage supply. The width of the pulse is determined by the time constant of an RC network, which consists of a capacitor (C) and a resistor (R). The pulse ends when the charge on the C equals 2/3 of the supply voltage. The pulse width can be lengthened or shortened to the need of the specific application by adjusting the values of R and C. The pulse width of time  $t$ , which is the time it takes to charge C to 2/3 of the supply voltage, is given by :

$$T = RC \ln(3) = 1.1 RC$$

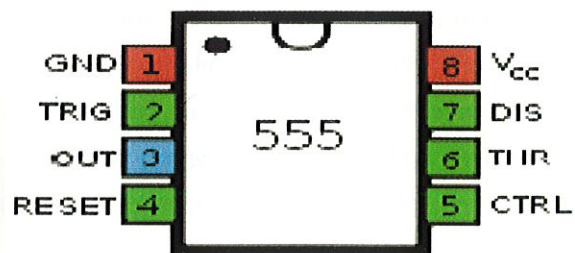
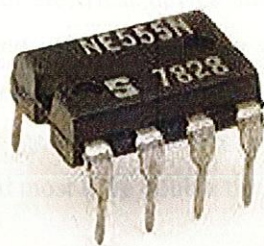
- ii. **Astable Mode or Free Running Mode:** The timer IC NE555 can operate as an oscillator. Uses include LED and lamp flashers, pulse generation, logic clocks, tone generation, security alarms, pulse position modulation (PPM), etc. In astable mode, the '555 timer' puts out a continuous stream of rectangular pulses having a specified frequency. Resistor  $R_1$  is connected between  $V_{CC}$  and the discharge pin (pin 7) and another resistor ( $R_2$ ) is connected between the discharge pin (pin 7), and the trigger (pin 2) and threshold (pin 6)



pins that share a common node. Hence the capacitor is charged through  $R_1$  and  $R_2$ , and discharged only through  $R_2$ , since pin 7 has low impedance to ground during output low intervals of the cycle, therefore discharging the capacitor. In the astable mode, the frequency of the pulse stream depends on the values of  $R_1$ ,  $R_2$  and  $C$  and is given by :

$$f = 1 / \ln(2) * C * (R_1 + 2R_2)$$

- iii. **Bistable Mode or Schmitt Trigger:** The timer IC NE555 can operate as a flip-flop, if the DIS pin is not connected and no capacitor is used. Uses include bounce free latched switches, etc. In bistable mode, the 555 timer acts as a basic flip-flop. The trigger and reset inputs (pins 2 and 4 respectively on a 555) are held high via pull-up resistors while the threshold input (pin 6) is simply grounded. Thus configured, pulling the trigger momentarily to ground acts as a 'set' and transitions the output pin (pin 3) to  $V_{cc}$  (high state). Pulling the reset input to ground acts as a 'reset' and transitions the output pin to ground (low state). No capacitors are required in a bistable configuration. Pin 8 ( $V_{cc}$ ) is, of course, tied to  $V_{cc}$  while pin 1 (Gnd) is grounded. Pins 5 and 7 (control and discharge) are left floating.



Sr. No.	Name of Pin	Purpose of Pin
1	GND	Providing the ground.
2	TRIG	A short pulse high to low to trigger the timer.
3	OUT	During the time interval, output stays at VCC.
4	RESET	The timing interval can be interrupted by applying reset pulse to low.
5	CTRL	Control voltage allows access to the internal voltage divider.
6	THR	Threshold at which the interval ends.
7	DIS	Connected to a capacitor whose discharge time will influence the timing interval.
8	VCC	The positive supply voltage which must be between 3 to 15 V.

### 5.5.2. RELAY

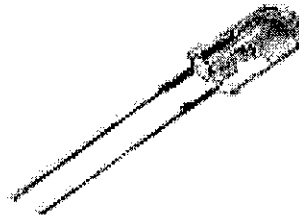
A relay is an electrical device such that the current flowing through it in one circuit can switch on and off a current in a second circuit. A relay is an electrically operated switch. Current flowing through the coil of the relay creates a magnetic field which attracts a lever and changes the switch contacts. The coil current can be on or off so relays have two switch positions and most have double throw (changeover) switch contacts as shown in the diagram.

Relays allow one circuit to switch a second circuit which can be completely separate from the first. For example a low voltage battery circuit can use a relay to switch a 230V AC mains circuit. There is no electrical connection inside the relay between the two circuits, the link is magnetic and mechanical.

### **5.5.3. IR LED**

Infrared Light Emitting Diode emits an infrared radiation. This radiation illuminates the surface in front of the IR LED. The surface reflects the infrared light and depending on the reflectivity of the surface, amount of light reflected varies. This reflected light is made incident on the reverse biased IR sensor.

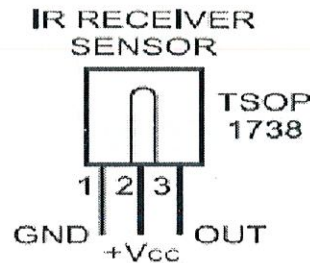
When photons are incident on the reverse biased junction of this diode, electron-hole pairs are generated, which results in reverse leakage current. Amount of electron-hole pairs generated depends on intensity of incident IR radiation. More intense radiation results in more reverse leakage current. This current can be passed through a resistor so as to get a proportional voltage. Thus, as the intensity of the incident rays varies, the voltage across the resistor will also vary accordingly.



### **5.5.4. IR RECEIVER SENSOR**

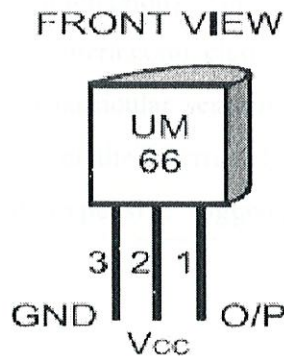
The TSOP 1738 infrared receiver sensor modulates the infrared signal and converts it into a corresponding electrical signal. TSOP 1738 is an infrared receiver sensor which is widely used in a large number of electronic products for receiving and demodulating infrared signals. These received demodulated signals can be easily decoded by a microcontroller. Many effective IR proximity sensors are built around the TSOP 1738 module. The TSOP module is commonly found at the receiving end of an IR remote control system. These modules require the incoming data to be modulated at a particular frequency and would ignore any other IR signals. It is also immune to ambient IR light, so one can easily use these sensors outdoors or under heavily lit conditions. Such modules are available for different carrier frequencies from 32 kHz to 42 kHz. In our project, we will be generating a constant stream of square wave signal using the IC NE555 centered at 38 kHz and would use it to drive

the IR led. So whenever this signal bounces off the obstacles, the receiver would detect it and change its output. Since the TSOP 1738 module works in the *active-low configuration*, its output would normally remain high and would go low when it detects the signal.



### 5.5.5. IC UM66

The IC UM66 will generate a music signal by taking a less voltage at its input side. We can use it in our commercial applications and also in designing our surveillance system. The simplest melody generator circuit can be made using an IC UM66. The UM66 series are CMOS ICs designed for using in a calling bell, phone and toys. It has a built in ROM programmed for playing the music. The device has a very low power consumption because of its CMOS technology. The melody (music / sound) will be available at pin3 of the IC UM66.



### 5.5.6. USB CABLING

USB, more formally known as the Universal Serial Bus is a specification to establish communication between devices and a host controller. Universal Serial Bus (USB) is a new external bus standard that supports data transfer rates of 12 Mbps (12 million

bits per second). A single USB port can be used to connect up to 127 peripheral devices, such as mice, modems, and keyboards. USB also supports plug-and-play installations. USB transfers isochronous or asynchronous data and is used to replace the cable clutter.



### **5.5.7. WEB CAMERA**

A webcam is a video capturing device connected to a computer or computer network, often using a USB port or, if connected to a network, using ethernet or Wi-Fi. The most popular use is for video telephony, permitting a computer to act as a videophone or video conferencing station. This can be used in messenger programs such as Windows Live Messenger, Skype and Yahoo messenger services. Other popular uses, which include the recording of video files or even still-images, are accessible via numerous software programs, applications and devices.

Web cameras are known for low manufacturing costs and flexibility, making them the lowest cost form of video telephony. The term 'webcam' may also be used in its original sense of a video camera connected to the web continuously for an indefinite time, rather than for a particular session, generally supplying a view for anyone who visits its web page over the Internet. Some of these, for example those used as online traffic cameras, are expensive, rugged professional video cameras.

## CHAPTER 6

### SOFTWARE COMPONENTS

#### 6.1. OVERVIEW

The software is been developed on .NET platform and the reason we choose .NET was because it provides us with many libraries which is already there and can be used as it is to perform a particular function.

Our software provides the 7 basic features. These are described below as:-

##### 6.1.1. VIDEO CAPTURING

Software has been made which uses Aforge libraries to capture the video from the webcam. The webcam on the other hand initiates the moment the IR rays are interrupted that means our video capturing also automatically initiates when the rays are cut.

##### 6.1.2. VIDEO SAVING

The same software interface lets you save the video which is being captured in the computer.

##### 6.1.3. IMAGE ACQUISITION

The code for the above interface which lets us capture and save the video stream also includes acquisition of images as it works on the principle of taking screen shots of the captured video, i.e. screen shots of the stream video was taken after a fixed time period automatically.

##### 6.1.4. IMAGE SAVING

The images captured or acquired from before are also saved in a folder in the computer. Although the images have been captured from the video stored but their small size is an advantage.



#### **6.1.5. IMAGE TRANSFERRING**

A different interface has been developed for transferring of images from the computer device on a mobile device using bluetooth. This creates a back up for the images just in case any harm happens to the computer.

#### **6.1.6. VIDEO TRANSFERRING**

The same code and interface gives transferring of video along with images using bluetooth technology. The process of creating backup is not automated as it dependent on bluetooth enabled devices present around the system and there can be more than one device around the system therefore we have to choose a device to send to it.

#### **6.1.7. SENDING WARNING SMS**

A another code has been made which sends a SMS to owner's mobile phone informing him/her about a possible intrusion made. This code also depends on bluetooth as the mobile on which the backup was made establishes a bluetooth connection with the mobile phone and then the SMS is sent through that phone to anther phone using GSM service network.

So as mentioned above our entire software can be broken into 3 major codes, as:

#### **Code 1: Video capturing, Video saving, Image Acquisition, Image saving.**

This code will be responsible for initiating video capturing and saving along with image capturing and saving.

For developing this code we have used Aforge libraries.

The predefined libraries of the Aforge Framework are described as follows and their various Classes, Interfaces, delegates, and the enumerations are described.

## AForge.Video namespace

The AForge.Video namespace contains interfaces and classes to access different video sources.

### Classes

Class	Description
1. JPEGStream	JPEG video source.
2. MJPEGStream	MJPEG video source.
3. NewFrameEventArgs	Arguments for new frame event from video source.
4. VideoSourceErrorEventArgs	Arguments for video source error event from video source.

### Interfaces

Interface	Description
1. IVideoSource	Video source interface.

### Delegates

Delegate	Description
1. NewFrameEventHandler	Delegate for new frame event handler.
2. PlayingFinishedEventHandler	Delegate for playing finished event handler.



- |    |                         |  |
|----|-------------------------|--|
| 3. | VideoSourceErrorHandler | Delegate for video source error event handler. |
|----|-------------------------|--|

**Enumerations**

- | Enumeration              | Description                        |
|--------------------------|------------------------------------|
| 1. ReasonToFinishPlaying | Reason of finishing video playing. |

**AForge.Video.DirectShow namespace**

The AForge.Video.DirectShow namespace contains classes, which allow to access video sources using DirectShow interface.

**Classes**

- | Class                   | Description   |
|-------------------------|---|
| 1. FileVideoSource      | Video source for video files.   |
| 2. FilterCategory       | DirectShow filter categories.   |
| 3. FilterInfo           | DirectShow filter information.  |
| 4. FilterInfoCollection | Collection of filters' information objects.                           |
| 5. VideoCapabilities    | Capabilities of video device such as frame size and frame rate.       |
| 6. VideoCaptureDevice   | Video source for local video capture device (for example USB webcam). |

**AForge.Video.VFW namespace**

The AForge.Video.VFW namespace contains classes, which allow reading and writing of AVI files using Video for Windows interface.

## Classes

	Class	Description
1.	AVIFileVideoSource	AVI file video source.
2.	AVIReader	AVI files reading using Video for Windows.
3.	AVIWriter	AVI files writing using Video for Windows interface.

Out of the mentioned function under Aforge we have used the following few as per requirement but the reason of making a note of them was to show the further scope of our project.

### Access to JPEG and MJPEG video streams

AForge.NET framework provides classes to access JPEG snapshots and MJPEG video stream over HTTP. This API is quite useful taking into account the fact, that most IP cameras support access to their video data through these simple protocols. Also usage of these classes is not only limited to IP cameras, since other sources may also provide JPEG snapshots and MJPEG video streams over HTTP.

#### JPEGStream class

This class represents the simplest way of getting access to continuously updating JPEG image over HTTP. Usually such updating JPEG images come from IP cameras, but any URL may be used to download image from there in loop. If the given URL points to static JPEG, the class will not be so useful. But once the URL points to CGI script (or any other server side application), the class will download updating JPEG images from there.

As an example, below code shows how to get video data (in form of consequent JPEG updates) from Axis cameras (see Axis documentation for their HTTP API):

```
// create JPEG video source

JPEGStream stream = new JPEGStream(
    "http://<axis_camera_ip>/axis-cgi/jpg/image.cgi" );

// set NewFrame event handler

stream.NewFrame += new NewFrameEventHandler( video_NewFrame );

// start the video source

stream.Start( );

// ...

// signal to stop

stream.SignalToStop( );

// ...

private void video_NewFrame( object sender,
    NewFrameEventArgs eventArgs )
{
    // get new frame

    Bitmap bitmap = eventArgs.Frame;

    // process the frame
}
```

### MJPEGStream class

The class has similar idea as the above one, but provides access not to JPEG images over HTTP, but to MJPEG video streams - streams, which are made of multiple JPEG images joint in sequence. Such types of video streams are usually provided by IP cameras however may be also provided by other services, which broadcast video to network using the simple protocol.

The below sample code also demonstrates accessing Axis camera, but now its MJPEG video stream:

```
// create MJPEG video source

MJPEGStream stream = new MJPEGStream(

    "http://<axis_camera_ip>/axis-cgi/mjpg/video.cgi" );

// set event handlers

stream.NewFrame += new NewFrameEventHandler( video_NewFrame );

// start the video source

stream.Start();

// ...

// signal to stop

stream.SignalToStop();
```

### Access to USB cameras and video files using DirectShow

AForge.NET framework provides classes to access USB web cameras and video files using DirectShow API. Since all the classes implement common interface, accessing USB camera is made as easy, as accessing video files or JPEG/MJPEG streams.

### VideoCaptureDevice class

This class allows getting access to different type of USB web cameras or other different devices, which support DirectShow interface. Below sample demonstrates accessing first available capture device in the system:

```
// enumerate video devices

videoDevices = new FilterInfoCollection(
    FilterCategory.VideoInputDevice );

// create video source

VideoCaptureDevice videoSource = new VideoCaptureDevice(
    videoDevices[0].MonikerString );

// set NewFrame event handler

videoSource.NewFrame += new NewFrameEventHandler( video_NewFrame );

// start the video source

videoSource.Start( );

// ...

// signal to stop

videoSource.SignalToStop( );

// ...

private void video_NewFrame( object sender,
```

```
NewFrameEventArgs eventArgs )  
  
{  
  
    // get new frame  
  
    Bitmap bitmap = eventArgs.Frame;  
  
    // process the frame  
  
}
```

#### FileVideoSource class

Using this class it is possible to play video files. All we need to change in the code is to create different video source class and, since it implements the same interface, all the rest will stay as before:

```
// create video source  
  
FileVideoSource videoSource = new FileVideoSource( fileName );  
  
// set NewFrame event handler  
  
videoSource.NewFrame += new NewFrameEventHandler( video_NewFrame );  
  
// start the video source  
  
videoSource.Start();  
  
// ...  
  
// signal to stop  
  
videoSource.SignalToStop();  
  
// ...
```

```

private void video_NewFrame( object sender,
    NewFrameEventArgs eventArgs )
{
    // get new frame
    Bitmap bitmap = eventArgs.Frame;

    // process the frame
}

```

### Reading/writing AVI files

AVIReader framework provides simple API to read and write AVI video files through Video for Windows interface. Although the interface is quite old and marked as obsolete by Microsoft, it is still supported and gives fairly simple API for accessing AVI video files.

#### AVIReader class

The class provides access to AVI video files and allows getting each video frame individually, as well as navigate through a video file specifying index of the next frame to receive:

```

// instantiate AVI reader
AVIReader reader = new AVIReader();

// open video file
reader.Open( "test.avi" );

```

```
// read the video file

while ( reader.Position - reader.Start < reader.Length )

{

    // get next frame

    Bitmap image = reader.GetNextFrame( );

    // .. process the frame somehow or display it

}

reader.Close( );
```

#### AVIWriter class

The class provides simple API for writing AVI video files. All you need to do is to specify codec to use, video size and frame rate and then start adding frames:

```
// instantiate AVI writer, use WMV3 codec

AVIWriter writer = new AVIWriter( "wmv3" );

// create new AVI file and open it

writer.Open( "test.avi", 320, 240 );

// create frame image

Bitmap image = new Bitmap( 320, 240 );

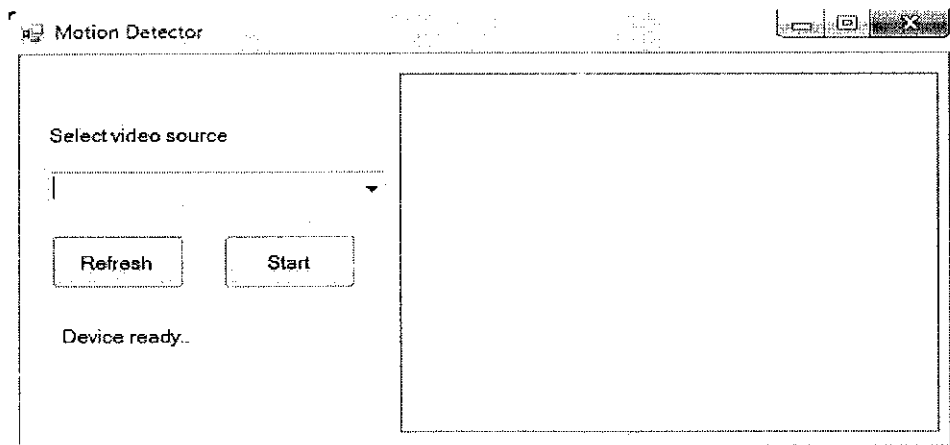
for ( int i = 0; i < 240; i++ )

{
```

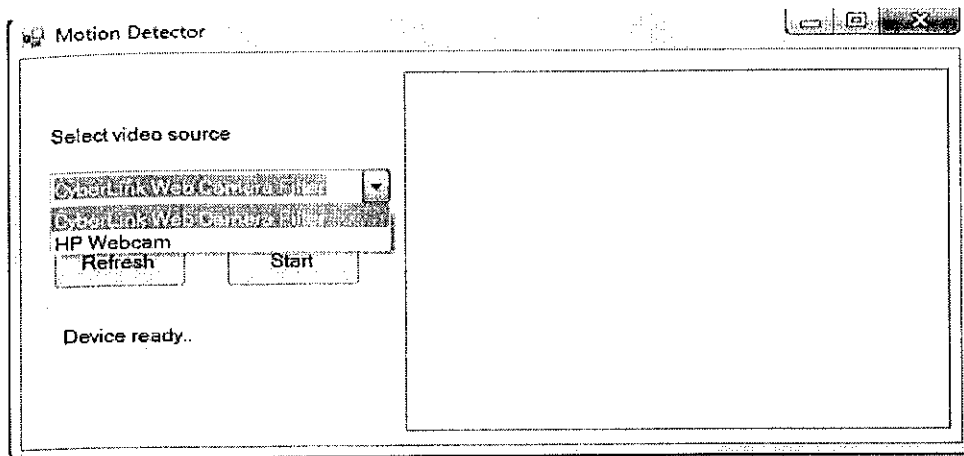


```
// update image  
image.SetPixel( i, i, Color.Red );  
  
// add the image as a new frame of video file  
writer.AddFrame( image );  
  
}  
  
writer.Close( );
```

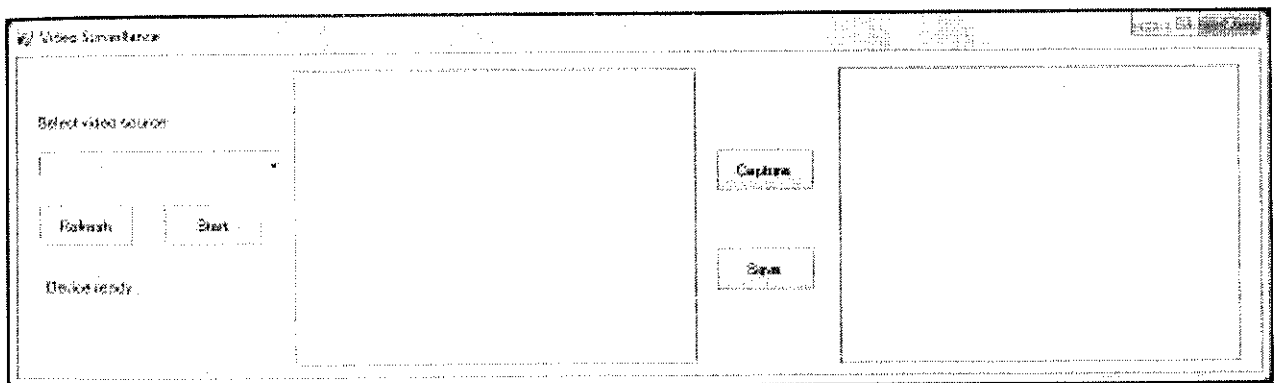
**The snapshots of the interface developed are shown below with a little description.**



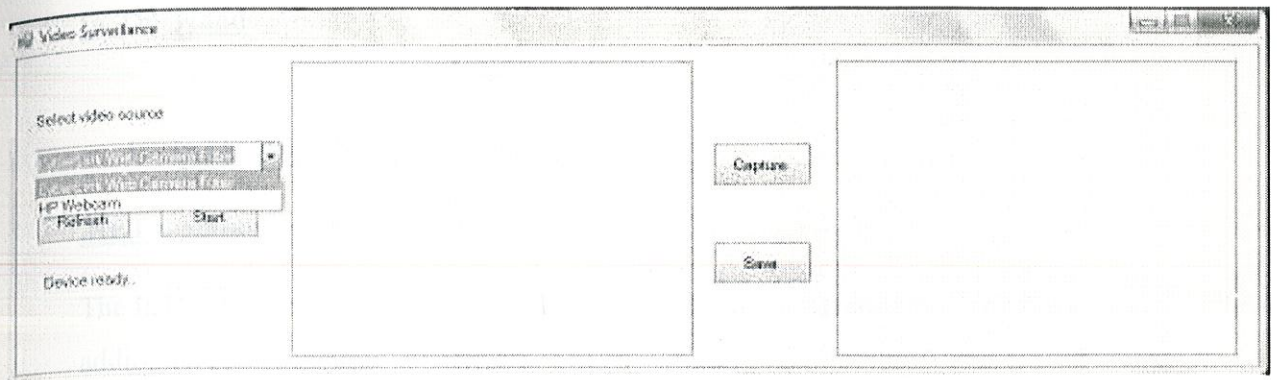
In the above software screenshot, we can see the functionality of the video streaming software such that is capable of selecting the video source and then by clicking the start button we can start the live video streaming.



In the above software screenshot, we can select the appropriate video source from the drop down menu and start the live video streaming from that particular device by clicking the start button.



In the above software screenshot, we can see the functionality of the video capturing software such that it is capable of capturing the currently streamed video by clicking the capture button and then we can also save that particular captured video by clicking on the save button. In case we do not find the captured video appropriate, then we can click on the capture button again to capture a new video which would meet our requirements.



In the above software screenshot, the functionality of the software is more or less the same as the previous screenshot, the only addition being the choice of the video source, which can be selected from the drop down menu.

**Code 2: For establishing a bluetooth connection with the mobile to transfer images and vidoes from master computer to master mobile phone for backup.**

So this code will find bluetooth enabled devices, establish the bluetooth connection with one of the found devices and then select the images and the video to be sent for backup and then finally send them.

This is actually socket programming and includes two major libraries, i.e. inthehand library and breham library.

The process involves first access of the system, which is password protected, so though proper authentication the administrator enters the system, switches on the bluetooth of the device, then start searching for the device, on finding and selecting one, ask for the files to be transfered and then finally sending them.

The predefined libraries of the Inthehand Framework and Braham framework are described as follows and their various Classes.

## In The Hand

### 1. InTheHand.Net Namespace

#### .NET Components for Mobility

The InTheHand.Net namespace provides a simple programming interface for additional protocols used on networks today.

#### Declaration Syntax

```
namespace InTheHand.Net
```

Icon	Type	Description
1.	BluetoothAddress	Represents a Bluetooth device address.
2.	BluetoothEndPoint	Establishes connections to a peer device and provides bluetooth port information.
3.	DnsEndPoint	Represents a network endpoint as a host name or a string representation of an IP address and a port number.
4.	DownloadDataCompletedEventArgs	Provides data for the DownloadDataCompleted event.
5.	DownloadDataCompletedEventHandler	Represents the method that will handle the DownloadDataCompleted event of a WebClient.
6.	DownloadProgressChangedEventArgs	Provides data for the DownloadProgressChanged event of a WebClient.
7.	DownloadProgressChangedEventHandler	Represents the method that will handle the DownloadProgressChanged event of

- a WebClient.
8. DownloadStringCompletedEventArgs Provides data for the DownloadStringCompleted event.
  9. DownloadStringCompletedEventHandler Represents the method that will handle the DownloadStringCompleted event of a WebClient.
  10. WebRequestMethods.File Represents the types of file protocol methods that can be used with a FILE request.
  11. FileWebRequest Provides a file system implementation of the WebRequest class.
  12. FileWebResponse Provides a file system implementation of the WebResponse class.
  13. WebRequestMethods.Ftp Represents the types of FTP protocol methods that can be used with an FTP request.
  14. FtpStatusCode Specifies the status codes returned for a File Transfer Protocol (FTP) operation.
  15. FtpWebRequest Implements a File Transfer Protocol (FTP) client.
  16. FtpWebResponse Encapsulates a File Transfer Protocol (FTP) server's response to a request.

17. `WebRequestMethods.Http` Represents the types of HTTP protocol methods that can be used with an HTTP request.
18. `IrDAAddress` Represents an IrDA device address.
19. `IrDAEndPoint` Represents an end point for an infrared connection.
20. `NetworkShare` Provides functionality for attaching to Windows networking shares.
21. `ObexListener` Provides a simple, programmatically controlled OBEX protocol listener.
22. `ObexListenerContext` Provides access to the request and response objects used by the `ObexListener` class.
23. `ObexListenerRequest` Describes an incoming OBEX request to an `ObexListener` object.
24. `ObexMethod` Methods which can be carried out in an Object Exchange transaction.
25. `ObexStatusCode` Specifies the status codes returned for an Object Exchange (OBEX) operation.
26. `ObexTransport` Supported network transports for Object Exchange.
27. `ObexWebRequest` Provides an OBEX implementation of the `WebRequest` class.

- |     |                                   |  |
|-----|-----------------------------------|--|
| 28. | ObexWebResponse                   | Provides an OBEX implementation of the WebResponse class.                              |
| 29. | UploadDataCompletedEventArgs      | Provides data for the UploadDataCompleted event.                                       |
| 30. | UploadDataCompletedEventHandler   | Represents the method that will handle the UploadDataCompleted event of a WebClient.   |
| 31. | UploadFileCompletedEventArgs      | Provides data for the UploadFileCompleted event.                                       |
| 32. | UploadFileCompletedEventHandler   | Represents the method that will handle the UploadFileCompleted event of a WebClient.   |
| 33. | UploadProgressChangedEventArgs    | Provides data for the UploadProgressChanged event of a WebClient.                      |
| 34. | UploadProgressChangedEventHandler | Represents the method that will handle the UploadProgressChanged event of a WebClient. |
| 35. | UploadStringCompletedEventArgs    | Provides data for the UploadStringCompleted event.                                     |
| 36. | UploadStringCompletedEventHandler | Represents the method that will handle the UploadStringCompleted event of a WebClient. |
| 37. | UploadValuesCompletedEventArgs    | Provides data for the UploadValuesCompleted event.                                     |

- |     |                                   |  |
|-----|-----------------------------------|--|
| 38. | UploadValuesCompletedEventHandler | Represents the method that will handle the UploadValuesCompleted event of a WebClient.                     |
| 39. | WebClient                         | Provides helper methods for sending data to and<br><br>Receiving data from a resource identified by a URL. |
| 40. | WebRequestMethods                 | Container class<br>for WebRequestMethods.Ftp<br><br>and WebRequestMethods.Http classes.                    |

## 2. InTheHand.Net.Bluetooth Namespace

### .NET Components for Mobility

The InTheHand.Net.Bluetooth namespace contains classes for working with Bluetooth functionality such as Radio hardware.

#### Declaration Syntax

namespace InTheHand.Net.Bluetooth

Icon	Type	Description
1.	AttributeIdLookup	Retrieves the name of the SDP Attribute ID with the given value in the specified Attribute ID class sets. Implementing Enum-like behaviour.
2.	BluetoothProtocolDescriptorType	Configures what type of element will be added by the ServiceRecordBuilder for theProtocolDescriptorList attribute.
3.	BluetoothPublicFactory	Provides the means to create Bluetooth classes on the one selected Bluetooth stack where multiple are



- loaded in the same process.
4. BluetoothRadio Represents a Bluetooth Radio device.
  5. BluetoothSecurity Handles security between bluetooth devices.
  6. BluetoothService Standard Bluetooth Profile identifiers.
  7. BluetoothWin32Authentication Provides Bluetooth authentication services on desktop Windows.
  8. BluetoothWin32AuthenticationEventArgs Provides data for an authentication event.
  9. ClassOfDevice Describes the device and service capabilities of a device.
  10. DeviceClass Class of Device flags as assigned in the Bluetooth specifications.
  11. ElementType Represents the types that an SDP element can hold.
  12. ElementTypeDescriptor Represents the type of the element in the SDP record binary format, and is stored as the higher 5 bits of the header byte.
  13. HardwareStatus Specifies the current status of the Bluetooth hardware.
  14. LanguageBaseItem Represents a member of the SDP LanguageBaseAttributeIdList, Attribute which provides for multi-language strings in a record.
  15. LinkPolicy Flags to describe Link Policy.
  16. Manufacturer Manufacturer codes.
  17. MapServiceClassToAttributeIdList Gets a list of enum-like classes containing SDP Service Attribute Id definitions for a particular

	Service Class.
18. RadioMode	Determine all the possible modes of operation of the Bluetooth radio.
19. ServiceAttribute	Holds an attribute from an SDP service record.
20. ServiceAttributeId	A Service Attribute Id identifies each attribute within an SDP service record.
21. ServiceClass	
22. ServiceElement	Holds an SDP data element.
23. ServiceRecord	Holds an SDP service record.
24. ServiceRecordBuilder	Provides a simple way to build a ServiceRecord, including ServiceClassIds and ServiceNames attributes etc.
25. ServiceRecordCreator	Creates a Service Record byte array from the given ServiceRecord object.
26. ServiceRecordHelper	Some useful methods for working with a SDP ServiceRecord including creating and accessing the ProtocolDescriptorList for an RFCOMM service.
27. ServiceRecordParser	Parses an array of bytes into the contained SDP ServiceRecord.
28. ServiceRecordUtilities	Utilities method working on SDP ServiceRecords, for instance to produce a 'dump' of the record's contents.
29. SizeIndex	Represents the size of the SDP element in the record binary format, and is stored as the lower 3 bits of the header byte.

30. `StringWithLanguageBaseAttribute` Indicates that the field to which it is applied represents an SDP Attribute that can exist in multiple language instances and thus has a language base offset applied to its numerical ID when added to a record.

### 3. InTheHand.Net.Sockets Namespace

#### .NET Components for Mobility

The `InTheHand.Net.Sockets` namespace provides added functionality for working with IrDA and Bluetooth Sockets.

#### Declaration Syntax

```
namespace InTheHand.Net.Sockets
```

Icon	Type	Description
	1. <code>AddressFamily32</code>	Specifies additional addressing schemes that an instance of the <code>Socket</code> class can use.
	2. <code>BluetoothClient</code>	Provides client connections for Bluetooth network services.
	3. <code>BluetoothDeviceInfo</code>	Provides information about an available device obtained by the client during device discovery.
	4. <code>BluetoothListener</code>	Listens for connections from Bluetooth network clients.
	5. <code>BluetoothProtocolType</code>	Specifies additional protocols that the <code>Socket</code> class supports.
	6. <code>BluetoothSocketOptionLevel</code>	Defines additional Bluetooth socket option levels for the <code>SetSocketOption(SocketOptionLevel, SocketOptionName,</code>

- Int32) and GetSocketOption(SocketOptionLevel, SocketOptionName)methods.
- 7. BluetoothSocketOptionName Defines Socket configuration option names for the Socket class.
  - 8. IrDACharacterSet Describes the character sets supported by the device.
  - 9. IrDAClient Makes connections to services on peer IrDA devices.
  - 10 IrDADeviceInfo Provides information about remote devices connected by infrared communications.
  - 11 IrDAHints Describes an enumeration of possible device types, such as Fax.
  - 12 IrDAListener Places a socket in a listening state to monitor infrared connections from a specified service or network address.
  - 13 IrDASocketOptionLevel Defines additional IrDA socket option levels for the SetSocketOption(SocketOptionLevel, SocketOptionName, Int32) and GetSocketOption(SocketOptionLevel, SocketOptionName)methods.
  - 14 IrDASocketOptionName Socket option constants to set IrDA specific connection modes, and get/set IrDA specific features.

#### 4. InTheHand.Windows.Forms Namespace

##### .NET Components for Mobility

The InTheHand.Windows.Forms namespace contains classes for creating Windows-based applications.

##### Declaration Syntax

```
namespace InTheHand.Windows.Forms
```

Icon	Type	Description
1.	ApplicationHelper	Provides properties which extend the Application class.
2.	AuthenticationDialog	The AuthenticationDialog class launches the Authentication dialog which prompts the user for a username password.
3.	BatteryChargeStatus	Defines identifiers that indicate the current battery charge level or charging state information.
4.	ButtonState	Specifies the appearance of a button.
5.	CaptionButton	Specifies the type of caption button to display
6.	ComboBoxHelper	Provides supporting methods for ComboBox.
7.	ControlHelper	Provides supporting methods for Control.
8.	ControlPaint	Provides methods used to paint common Windows controls and their elements.
9.	CreateParams	Encapsulates the information needed when creating a control.
10.	CustomListView	An enhanced ListView which supports custom

- drawing of individual ListViewItem.
11. DrawListViewItemEventArgs Provides data for the DrawItem event.
  12. DrawListViewItemEventHandler Represents the method that will handle the DrawItem event of a CustomListView.
  13. Help Provides Help support for all Windows Mobile platforms.
  14. IMessageFilter Defines a message filter interface.
  15. ItemBoundsPortion Specifies a portion of the list view item from which to retrieve the bounding rectangle.
  16. ListBoxHelper Provides supporting methods for ListBox.
  17. ListViewHelper Provides supporting methods for ListView.
  18. ListViewItemStates Defines constants that represent the possible states of a ListViewItem.
  19. MethodInvoker Represents a delegate that can execute any method in managed code that is declared void and takes no parameters.
  20. NativeWindow Provides a low-level encapsulation of a window handle and a window procedure.
  21. PowerLineStatus Specifies the system power status.
  22. PowerStatus Indicates current system power status information.
  23. ScrollButton Specifies the type of scroll arrow to draw on a scroll bar.
  24. SelectBluetoothDeviceDialog Provides a form to select an available Bluetooth device.
  25. SendKeys Provides methods for sending keystrokes to an

	application.
26. SystemInformationHelper	Provides information about the current system environment.
27. TabControlHelper	Provides helper methods for the TabControl on Windows Mobile 6.5.
28. TextBoxHelper	Provides supporting methods for TextBox.

## **Brecham**

### **Brecham.Obex — An OBEX library for the .NET Framework**

#### **Brecham.Obex Namespace**

##### Classes

Icon Class	Description
1. AbortableStream	An abstract Stream containing the functionality common to the ObexPutStream and ObexGetStream .
2. ObexClientSession	A client-side connection to an OBEX server, supports Put, Get, and most other operation types.
3. ObexConstant	Well-known values used with the OBEX protocol. These include values for use in the Target or Who headers to select which service/application to access, values for use in the Type to indicate the media type of

the content, and other miscellaneous values. The Target and Type values are defined in ObexConstant.Target and ObexConstant.Type respectively.

4. ObexConstant.Target Well-known values for use in the Target header usually at Connect time. They select which service or application is required, for instance the default Inbox service, or the Folder Browsing service for instance.
5. ObexConstant.Type Well-known Media Type values defined (or at least referenced) in the OBEX specification.
6. ObexGetStream Provides the stream of content in a OBEX Get operation.
7. ObexHeaderCollection Contains OBEX protocol headers associated with a request or response.
8. ObexHeaderConverter Methods to convert types into the formats required by various OBEX header types.
9. ObexPutStream Provides the stream of content in a OBEX Put operation.
10. ObexResponseException The exception that is thrown when an error occurs while accessing the network through the OBEX protocol.

### Enumerations

icon	Enumeration	Description
	1. BackupFirst	Indicate whether on a SETPATH operation that the server should should first change to the parent folder,



before moving to the named folder etc.

2. IfFolderDoesNotExist Indicate whether on a SETPATH operation that a new folder should be created by the server if it doesn't already exist, or whether it should instead return an error in that case.
3. ObexHeaderId OBEX Header Identifiers.

### File transfer

We will present a program that lets you browse any device connected to your computer via Bluetooth and allows you to upload/download files to/from the device. The device should have OBEX support. In order to connect to a device via Bluetooth and to perform OBEX operations, we used these libraries:[32feet.Net](#) and [Brecham](#)  
OBEX

### Requirements

In order for this application to function, you need to have Bluetooth on your computer that uses the Microsoft Bluetooth stack and another device with Bluetooth which you will connect to use this program. If your Bluetooth device uses a non-Microsoft stack, then it is possible to disable it and install the Microsoft stack. Have

This program uses the OBEX library that communicates to a device, so a general understanding of what OBEX is and how it works is preferable, but not desired.

### How the Application Works

#### Connecting

When you run the application, the first thing you should do is connect to the device. You can choose the device you want to connect to using a dialog that shows the available Bluetooth devices. After the device has been selected, we connect to it and initiate a new session. The code snippet shows how it is done:

```

private void Connect()
{
    using (SelectBluetoothDeviceDialog bldialog =
        new SelectBluetoothDeviceDialog())
    {
        bldialog.ShowAuthenticated = true;
        bldialog.ShowRemembered = true;
        bldialog.ShowUnknown = true;

        if (bldialog.ShowDialog() == DialogResult.OK)
        {
            if (bldialog.SelectedDevice == null)
            {
                MessageBox.Show("No device selected", "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);

                return;
            }

            //Create new end point for the selected device.

            //BluetoothService.ObexFileTransfer means
            //that we want to connect to Obex service.

            BluetoothDeviceInfo selecteddevice = bldialog.SelectedDevice;

```

```

BluetoothEndPoint remoteEndPoint = new
BluetoothEndPoint(selecteddevice.DeviceAddress,
BluetoothService.ObexFileTransfer);

//Create new Bluetooth client..
client = new BluetoothClient();
try
{
    //... and connect to the end point we created.
    client.Connect(remoteEndPoint);

    //Create a new instance of ObexClientSession
    session = new ObexClientSession(client.GetStream(), UInt16.MaxValue);
    session.Connect(ObexConstant.Target.FolderBrowsing);
}
catch (SocketException ex)
{
    ExceptionHandler(ex, false);
    return;
}
catch (ObjectDisposedException ex)
{

```

```

        ExceptionHandler(ex, false);

        return;
    }

    catch (IOException ex)
    {
        ExceptionHandler(ex, false);

        return;
    }

    bgwWorker.RunWorkerAsync();
}
}
}

```

First, we show a dialog that displays the available Bluetooth devices. In addition to the devices that are present at the moment, it will also show those that were connected to the computer in the past but might not be available now. This can be turned off by setting the `ShowRemembered` property of `SelectBluetoothDeviceDialog` to `false`. However, in that case, if you want to connect to a remembered device, it will not be shown by the dialog.

After the device has been selected, we create a remote end point based on the address of the device. The second parameter specifies the service we want to connect to. In our case, it is `BluetoothService.ObexFileTransfer`, meaning that we will be able to transfer files using the OBEX protocol. Next, we need to create an instance of the `BluetoothClient` class and connect to the end point we created earlier. When the connection has been established, we create an instance of the `ObexClientSession` class. According to documentation, "[ObexClientSession is] A

client-side connection to an OBEX server, supports Put, Get, and most other operation types." The instance we create will be used for all the OBEX operations we will perform. Next, we connect to the folder browsing service so that we can browse the device using OBEX.

Now, when we are connected to the folder browsing service of the device, we can start exploring it. We will be able to show the files and folders, create new folders, delete existing ones, and refresh folder content.

### Downloading and Uploading Files

In order to download or upload files, you can use the GetTo or PutFrom methods. However, to report progress, you will need to create a new stream type and use it in conjunction with the Decorator Pattern. A simpler way for progress reporting is to use the Get and Put methods. Both of them return a Stream object. In the case of downloading, we should read from the returned stream and write to the FileStream object, and in the case of uploading, we should read from the FileStream and write to the stream returned by the Put method. In both cases, we can count how many bytes we have read and report progress depending on it. This is done using the BackgroundWorker too.

```
private void bgwWorker_DoWork(object sender, DoWorkEventArgs e)
{
    long progress = 0;

    DateTime start = DateTime.Now;

    for (int i = 0; i < filesToProcess.Count; i++)
```

```

{
    string currentfile = filesToProcess[i];

    //Report that we started downloading new file
    bgwWorker.ReportProgress((int)(((progress * 100) / totalsize)), i + 1);

    string filename = download ? Path.Combine(dir, currentfile) : currentfile;

    //Stream on our file system. We will need to either read from it or write to it.
    FileStream hoststream = download ?
        new FileStream(filename, FileMode.Create, FileAccess.Write, FileShare.None)
        : new FileStream(filename, FileMode.Open, FileAccess.Read, FileShare.None);

    AbortableStream remotestream = null;

    try
    {
        //Stream on our device. We will need to either read from it or write to it.
        remotestream = download ? (AbortableStream)currentSession.Get(currentfile,
null)
        : (AbortableStream)currentSession.Put(Path.GetFileName(currentfile), null);
    }

    catch (IOException ex)
    {
        exceptionoccured = true;

        ExceptionMethod(ex);

        return;
    }
}

```

```

catch (ObexResponseException ex)
{
    exceptionoccured = true;

    ExceptionMethod(ex);

    return;
}

using (hoststream)
{
    using (remotestream)
    {
        //This is the function that does actual reading/writing.

        long result = download ?

            ProcessStreams(remotestream, hoststream, progress, currentfile)

            :ProcessStreams(hoststream, remotestream, progress, currentfile);

        if (result == 0)
        {
            e.Cancel = true;

            //Even if we are cancelled we need to report how many files we have already

            //uploaded so that they are added to the listview. Or if it is download we

            //need to delete the partially downloaded last file.

            filesProcessed = i;

            return;
        }
    }
}

```

```

    }
else
    progress = result;
}
}
}

DateTime end = DateTime.Now;

e.Result = end - start;
}

```

As the process is similar in both cases, there is one function that does the actual work. The function reads from the source stream and writes to the destination stream. This is how it works:

```

private long ProcessStreams(Stream source, Stream destination, long progress,
                            string filename)
{
    //Allocate buffer
    byte[] buffer = new byte[1024 * 4];

    while (true)
    {
        //Report downloaded file size
        bgwWorker.ReportProgress((int)((((progress * 100) / totalsize)), progress);

        if (bgwWorker.CancellationPending)
        {

```



```
currentSession.Abort();

return 0;

}

try

{

    //Read from source and write to destination.

    //Break if finished reading. Count read bytes.

    int length = source.Read(buffer, 0, buffer.Length);

    if (length == 0) break;

    destination.Write(buffer, 0, length);

    progress += length;

}

//Return 0 as if operation was cancelled so that processedFiles is set.

catch (IOException ex)

{

    exceptionoccured = true;

    ExceptionMethod(ex);

    return 0;

}

catch (ObexResponseException ex)

{
```

```
exceptionoccured = true;

ExceptionMethod(ex);

return 0;

}

}

return progress;

}
```

**Code 3: For establishing a bluetooth connection with the mobile to send intrusion warning SMS from the connected master mobile phone to owner's mobile phone using GSM service network with normal charges charged.**

This code includes establishing a bluetooth connection with the master mobile phone and then sending SMS through that phone to the owner phone. This is automatic process and the message content of the message can be modified as per need. It informs the owner of a possible intrusion in their secure area.

It uses AT commands for the process the bluetooth connection has already been established by the previous code. It makes use of that connection and send a series of command to the master mobile phone for creating a SMS and then sending it automatically.

In general, there are two ways to send SMS messages from a computer / PC to a mobile phone:

Connect a mobile phone or GSM/GPRS modem to a computer / PC. Then use the computer / PC and AT commands to instruct the mobile phone or GSM/GPRS modem to send SMS messages.

Connect the computer / PC to the SMS center (SMSC) or SMS gateway of a wireless carrier or SMS service provider. Then send SMS messages using a protocol / interface supported by the SMSC or SMS gateway.

### **The 1st Way: Sending SMS Messages from a Computer Using a Mobile Phone or GSM/GPRS Modem**

The SMS specification has defined a way for a computer to send SMS messages through a mobile phone or GSM/GPRS modem. A GSM/GPRS modem is a wireless modem that works with GSM/GPRS wireless networks. A wireless modem is similar to a dial-up modem. The main difference is that a wireless modem transmits data through a wireless network whereas a dial-up modem transmits data through a copper telephone line. More information about GSM/GPRS modems will be provided in the section "Introduction to GSM / GPRS Wireless Modems". Most mobile phones can be used as a wireless modem. However, some mobile phones have certain limitations comparing to GSM/GPRS modems. This will be discussed in the section "Which is Better: Mobile Phone or GSM / GPRS Modem" later.

To send SMS messages, first place a valid SIM card from a wireless carrier into a mobile phone or GSM/GPRS modem, which is then connected to a computer. There are several ways to connect a mobile phone or GSM/GPRS modem to a computer. For example, they can be connected through a serial cable, a USB cable, a Bluetooth link or an infrared link. The actual way to use depends on the capability of the mobile phone or GSM/GPRS modem. For example, if a mobile phone does not support Bluetooth, it cannot connect to the computer through a Bluetooth link.

After connecting a mobile phone or GSM/GPRS modem to a computer, you can control the mobile phone or GSM/GPRS modem by sending instructions to it. The instructions used for controlling the mobile phone or GSM/GPRS modem are called AT commands. (AT commands are also used to control dial-up modems for wired telephone system.) Dial-up modems, mobile phones and GSM/GPRS modems support a common set of standard AT commands. In addition to this common set of standard

AT commands, mobile phones and GSM/GPRS modems support an extended set of AT commands. One use of the extended AT commands is to control the sending and receiving of SMS messages.

The following table lists the AT commands that are related to the writing and sending of SMS messages:

AT command	Meaning
+CMGS	Send message
+CMSS	Send message from storage
+CMGW	Write message to memory
+CMGD	Delete message
+CMGC	Send command
+CMMS	More messages to send

One way to send AT commands to a mobile phone or GSM/GPRS modem is to use a terminal program. A terminal program's function is like this: It sends the characters you typed to the mobile phone or GSM/GPRS modem. It then displays the response it receives from the mobile phone or GSM/GPRS modem on the screen. The terminal program on Microsoft Windows is called HyperTerminal. More details about the use of Microsoft HyperTerminal can be found in the "[How to Use Microsoft HyperTerminal to Send AT Commands to a Mobile Phone or GSM/GPRS Modem](#)" section of this SMS tutorial.

Below shows a simple example that demonstrates how to use AT commands and the HyperTerminal program of Microsoft Windows to send an SMS text message. The lines in bold type are the command lines that should be entered in HyperTerminal.

The other lines are responses returned from the GSM / GPRS modem or mobile phone.

AT

OK

AT+CMGF=1

OK

AT+CMGW="+85291234567"

> A simple demo of SMS text messaging.

+CMGW: 1

OK

AT+CMSS=1

+CMSS: 20

OK

Here is a description of what is done in the above example:

Line 1: "AT" is sent to the GSM / GPRS modem to test the connection. The GSM / GPRS modem sends back the result code "OK" (line 2), which means the connection between the HyperTerminal program and the GSM / GPRS modem works fine.

Line 3: The AT command +CMGF is used to instruct the GSM / GPRS modem to operate in SMS text mode. The result code "OK" is returned (line 4), which indicates the command line "AT+CMGF=1" has been executed successfully. If the result code "ERROR" is returned, it is likely that the GSM / GPRS modem does not support the SMS text mode. To confirm, type "AT+CMGF=?" in the HyperTerminal program. If the response is "+CMGF: (0,1)" (0=PDU mode and 1=text mode), then SMS text mode is supported. If the response is "+CMGF: (0)", then SMS text mode is not supported.

Line 5 and 6: The AT command +CMGW is used to write an SMS text message to the message storage of the GSM / GPRS modem. "+85291234567" is the recipient mobile

phone number. After typing the recipient mobile phone number, you should press the Enter button of the keyboard. The GSM / GPRS modem will then return a prompt ">" and you can start typing the SMS text message "A simple demo of SMS text messaging.". When finished, press Ctrl+z of the keyboard.

Line 7: "+CMGW: 1" tells us that the index assigned to the SMS text message is 1. It indicates the location of the SMS text message in the message storage.

Line 9: The result code "OK" indicates the execution of the AT command +CMGW is successful.

Line 10: The AT command +CMSS is used to send the SMS text message from the message storage of the GSM / GPRS modem. "1" is the index of the SMS text message obtained from line 7.

Line 11: "+CMSS: 20" tells us that the reference number assigned to the SMS text message is 20.

Line 13: The result code "OK" indicates the execution of the AT command +CMSS is successful.

To send SMS messages from an application, you have to write the source code for connecting to and sending AT commands to the mobile phone or GSM/GPRS modem, just like what a terminal program does. You can write the source code in C, C++, Java, Visual Basic, Delphi or other programming languages you like. However, writing your own code has a few disadvantages:

You have to learn how to use AT commands.

You have to learn how to compose the bits and bytes of an SMS message. For example, to specify the character encoding (e.g. 7-bit encoding and 16-bit Unicode encoding) of an SMS message, you need to know which bits in the message header should be modified and what value should be assigned.

Sending SMS messages with a mobile phone or GSM/GPRS modem has a drawback - the SMS transmission speed is low. As your SMS messaging application becomes more popular, it has to handle a larger amount of SMS traffic and finally the mobile

phone or GSM/GPRS modem will not be able to take the load. To obtain a high SMS transmission speed, a direct connection to an SMSC or SMS gateway of a wireless carrier or SMS service provider is needed. However, AT commands are not used for communicating with an SMS center or SMS gateway. This means you have to make a big change to your SMS messaging application in order to move from a wireless-modem-based solution to a SMSC-based solution.

In most cases, instead of writing your own code for interacting with the mobile phone or GSM/GPRS modem via AT commands, a better solution is to use a high-level SMS messaging API (Application programming interface) / SDK (Software development kit) / library. The API / SDK / library encapsulates the low-level details. So, an SMS application developer does not need to know AT commands and the composition of SMS messages in the bit-level. Some SMS messaging APIs / SDKs / libraries support SMSC protocols in addition to AT commands. To move from a wireless-modem-based SMS solution to a SMSC-based SMS solution, usually you just need to modify a configuration file / property file or make a few changes to your SMS messaging application's source code.

Another way to hide the low-level AT command layer is to place an SMS gateway between the SMS messaging application and the mobile phone or GSM/GPRS modem. (This has been described in the section "What is an SMS Gateway?" earlier.) Simple protocols such as HTTP / HTTPS can then be used for sending SMS messages in the application. If an SMSC protocol (e.g. SMPP, CIMD, etc) is used for communicating with the SMS gateway instead of HTTP / HTTPS, an SMS messaging API / SDK / library can be very helpful to you since it encapsulates the SMSC protocol's details.

Usually a list of supported / unsupported mobile phones or wireless modems is provided on the web site of an SMS messaging API / SDK / library or an SMS gateway software package. Remember to check the list if you are going to use an SMS messaging API / SDK / library or an SMS gateway software package.

Major Drawback of Sending SMS Messages through a Mobile Phone or GSM/GPRS Modem -- Low SMS Sending Rate

Using a mobile phone or GSM/GPRS modem to send SMS messages has a major drawback that is the SMS sending rate is too low. Only 6-10 SMS messages can be sent per minute (when the "SMS over GSM" mode is used). The performance is not affected by the connection between the computer and the mobile phone or GSM/GPRS modem (i.e. the SMS sending rate is about the same no matter the mobile phone or GSM/GPRS modem is connected to the computer through a serial cable, USB cable, Bluetooth link or infrared link) and does not depend on whether a mobile phone or GSM/GPRS modem is used (i.e. the SMS sending rate is about the same no matter a mobile phone or a GSM/GPRS modem is used). The determining factor for the SMS sending rate is the wireless network.

### **The 2nd Way: Sending SMS Messages from a Computer through a Connection to the SMSC or SMS Gateway of a Wireless Carrier or SMS Service Provider**

The way for sending SMS messages from a computer through a mobile phone or GSM/GPRS modem has a major limitation that is the SMS sending rate is too low. If you need a high SMS sending rate, obtaining a direct connection to the SMS center (SMSC) or SMS gateway of a wireless carrier is necessary. The connection may be made through the Internet, X.25 or dial-up. If you cannot get a direct connection to the SMSC or SMS gateway of a wireless carrier, another choice is to get a connection to the SMS gateway of an SMS service provider, which will forward SMS messages towards a suitable SMSC.

### **Difficulties in Getting a Direct Connection to the SMSC or SMS Gateway of a Wireless Carrier**

It can be difficult for small businesses or individual application developers to obtain a direct connection to the SMSC or SMS gateway of a wireless carrier since a wireless carrier may only provide such service to those who have huge SMS traffic. Buying SMS messages in bulk means the total fee will be very high (although the fee per SMS message will be low).

Besides, the information about the service (for example, cost of the service, protocols supported, network coverage) is usually not stated clearly on a wireless carrier's web site. This is because the wireless carrier staff wants to know more about your SMS



messaging application, such as its nature and traffic requirement, before offering a price and providing further information to you. To decide which wireless carrier's service plan is the best, you have to discuss with the sales staff of each wireless carriers. This is troublesome if you just want to send a small number of SMS messages. (Of course if you need to send a large amount of SMS messages, say one million SMS messages per month, negotiating with the wireless carrier staff for a more favorable agreement is a necessary step.)

A more convenient way to send SMS messages is to use the SMS connectivity service of an SMS service provider.

SMS Service Providers (SMS Gateway Providers, SMS Resellers, SMS Brokers)

There is a demand for SMS connectivity from applications that does not require the sending or receiving of large amount of SMS messages. One example is a remote monitoring system. If the remote monitoring system finds that a certain server is not responding, it will send an SMS alert to the system administrator's mobile phone. This remote monitoring system will have a very small amount of SMS traffic per month since the servers being monitored should be working fine most of the time.

Since a wireless carrier usually does not provide direct SMSC or SMS gateway access to users without a large amount of SMS traffic, some companies come out to fill the gap. These companies are called SMS service providers. There is no minimal purchase requirement or monthly minimum usage requirement for many SMS service providers.

SMS service providers are also known as SMS gateway providers, SMS resellers and SMS brokers because of the following reasons:

SMS gateway providers -- An SMS service provider provides an SMS gateway for its users to send SMS messages to. This SMS gateway will then route the SMS messages to another SMS gateway or SMSC.

SMS resellers and SMS brokers -- SMS service providers buy a large amount of SMS messages from a lot of wireless carriers at a low price per SMS message. They then sell the SMS messages at a price higher than the cost.

Unlike wireless carriers, many SMS service providers provide detail information about their SMS connectivity service on their web site. For example, you may find the cost of the service, network coverage, protocols supported, developers' guide, etc, on the web site. Thus, the service of different SMS service providers can be compared easily. If you are not happy with, say the price or network coverage of an SMS service provider, you can simply leave its web site and find another SMS service provider.

Another advantage of using the SMS connectivity services of SMS service providers is that their network coverage is very good. They work hard to cover as many wireless networks as possible so as to make their services attractive. Some SMS service providers can send SMS messages not only to GSM wireless networks, but also to CDMA and TDMA wireless networks.

It is easy to send SMS messages with an SMS service provider. Here are the typical steps:

Register for an account on the SMS service provider's web site. (An SMS service provider may allow newly registered users to send a few free SMS messages for testing its service quality.)

Log into the account.

Buy a number of credits or SMS text messages online. Many SMS service providers support credit card payment and some also support PayPal.

Send SMS messages using a protocol / interface (e.g. HTTP, email, FTP) supported by the SMS service provider.

SMS service providers can be divided into two categories depending on how they require you to pay for their SMS messaging service:

Credit-based -- You purchase a number of credits from the SMS service provider's web site. Sending one SMS message will cost you one or more credits, depending on the country you send the SMS message to. For example, sending an SMS text message to India might cost you one credit while sending an SMS text message to the US might cost you two credits.

SMS-based -- You purchase a number of SMS messages from the SMS service provider's web site. The cost per SMS message is the same for all destinations. For example, if you purchase ten SMS messages, you can send at most ten SMS messages no matter the destination is India or the US.

The cost per SMS message sent depends on which SMS service provider you choose and how many SMS messages you purchase. It starts at around US \$0.06 to US \$0.07 per SMS message. The more SMS messages you purchased, the lower the cost of sending one SMS message.

### Introduction to AT Commands

AT commands are instructions used to control a modem. AT is the abbreviation of ATtention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands. Many of the commands that are used to control wired dial-up modems, such as ATD (Dial), ATA (Answer), ATH (Hook control) and ATO (Return to online data state), are also supported by GSM/GPRS modems and mobile phones. Besides this common AT command set, GSM/GPRS modems and mobile phones support an AT command set that is specific to the GSM technology, which includes SMS-related commands like AT+CMGS (Send SMS message), AT+CMSS (Send SMS message from storage), AT+CMGL (List SMS messages) and AT+CMGR (Read SMS messages).

Note that the starting "AT" is the prefix that informs the modem about the start of a command line. It is not part of the AT command name. For example, D is the actual AT command name in ATD and +CMGS is the actual AT command name in AT+CMGS. However, some books and web sites use them interchangeably as the name of an AT command.

Here are some of the tasks that can be done using AT commands with a GSM/GPRS modem or mobile phone:

Get basic information about the mobile phone or GSM/GPRS modem. For example, name of manufacturer (AT+CGMI), model number (AT+CGMM), IMEI number (International Mobile Equipment Identity) (AT+CGSN) and software version (AT+CGMR).

Get basic information about the subscriber. For example, MSISDN (AT+CNUM) and IMSI number (International Mobile Subscriber Identity) (AT+CIMI).

Get the current status of the mobile phone or GSM/GPRS modem. For example, mobile phone activity status (AT+CPAS), mobile network registration status (AT+CREG), radio signal strength (AT+CSQ), battery charge level and battery charging status (AT+CBC).

Establish a data connection or voice connection to a remote modem (ATD, ATA, etc).

Send and receive fax (ATD, ATA, AT+F\*).

Send (AT+CMGS, AT+CMSS), read (AT+CMGR, AT+CMGL), write (AT+CMGW) or delete (AT+CMGD) SMS messages and obtain notifications of newly received SMS messages (AT+CNMI).

Read (AT+CPBR), write (AT+CPBW) or search (AT+CPBF) phonebook entries.

Perform security-related tasks, such as opening or closing facility locks (AT+CLCK), checking whether a facility is locked (AT+CLCK) and changing passwords (AT+CPWD).

(Facility lock examples: SIM lock [a password must be given to the SIM card every time the mobile phone is switched on] and PH-SIM lock [a certain SIM card is associated with the mobile phone. To use other SIM cards with the mobile phone, a password must be entered.])

Control the presentation of result codes / error messages of AT commands. For example, you can control whether to enable certain error messages (AT+CMEE) and whether error messages should be displayed in numeric format or verbose format (AT+CMEE=1 or AT+CMEE=2).

Get or change the configurations of the mobile phone or GSM/GPRS modem. For example, change the GSM network (AT+COPS), bearer service type (AT+CBST), radio link protocol parameters (AT+CRLP), SMS center address (AT+CSCA) and storage of SMS messages (AT+CPMS).

Save and restore configurations of the mobile phone or GSM/GPRS modem. For example, save (AT+CSAS) and restore (AT+CRES) settings related to SMS messaging such as the SMS center address.

Note that mobile phone manufacturers usually do not implement all AT commands, command parameters and parameter values in their mobile phones. Also, the behavior of the implemented AT commands may be different from that defined in the standard. In general, GSM/GPRS modems designed for wireless applications have better support of AT commands than ordinary mobile phones.

In addition, some AT commands require the support of mobile network operators. For example, SMS over GPRS can be enabled on some GPRS mobile phones and GPRS modems with the +CGSMS command (command name in text: Select Service for MO SMS Messages). But if the mobile network operator does not support the transmission of SMS over GPRS, you cannot use this feature.

#### Basic Commands and Extended Commands

There are two types of AT commands: basic commands and extended commands.

Basic commands are AT commands that do not start with "+". For example, D (Dial), A (Answer), H (Hook control) and O (Return to online data state) are basic commands.

Extended commands are AT commands that start with "+". All GSM AT commands are extended commands. For example, +CMGS (Send SMS message), +CMSS (Send SMS message from storage), +CMGL (List SMS messages) and +CMGR (Read SMS messages) are extended commands.

## General Syntax of Extended AT Commands

The general syntax of extended AT commands is straightforward. The syntax rules are provided below. The syntax of basic AT commands is slightly different. We will not cover the syntax of basic AT commands in this SMS tutorial since all SMS messaging commands are extended AT commands.

Syntax rule 1. All command lines must start with "AT" and end with a carriage return character. (We will use <CR> to represent a carriage return character in this SMS tutorial.) In a terminal program like HyperTerminal of Microsoft Windows, you can press the Enter key on the keyboard to output a carriage return character.

Example: To list all unread inbound SMS messages stored in the message storage area, type "AT", then the extended AT command "+CMGL", and finally a carriage return character, like this:

```
AT+CMGL<CR>
```

Syntax rule 2. A command line can contain more than one AT command. Only the first AT command should be prefixed with "AT". AT commands in the same command-line string should be separated with semicolons.

Example: To list all unread inbound SMS messages stored in the message storage area and obtain the manufacturer name of the mobile device, type "AT", then the extended AT command "+CMGL", followed by a semicolon and the next extended AT command "+CGMI":

```
AT+CMGL;+CGMI<CR>
```

An error will occur if both AT commands are prefixed with "AT", like this:

```
AT+CMGL;AT+CGMI<CR>
```

Syntax rule 3. A string is enclosed between double quotes.

Example: To read all SMS messages from message storage in SMS text mode (at this time you do not need to know what SMS text mode is. More information will be provided later in this SMS tutorial), you need to assign the string "ALL" to the extended AT command +CMGL, like this:

```
AT+CMGL="ALL"<CR>
```

Syntax rule 4. Information responses and result codes (including both final result codes and unsolicited result codes) always start and end with a carriage return character and a linefeed character.

Example: After sending the command line "AT+CGMI<CR>" to the mobile device, the mobile device should return a response similar to this:

```
<CR><LF>Nokia<CR><LF>
```

```
<CR><LF>OK<CR><LF>
```

The first line is the information response of the AT command +CGMI and the second line is the final result code. <CR> and <LF> represent a carriage return character and a linefeed character respectively. The final result code "OK" marks the end of the response. It indicates no more data will be sent from the mobile device to the computer / PC.

When a terminal program such as HyperTerminal of Microsoft Windows sees a carriage return character, it moves the cursor to the beginning of the current line. When it sees a linefeed character, it moves the cursor to the same position on the next line. Hence, the command line "AT+CGMI<CR>" that you entered and the

corresponding response will be displayed like this in a terminal program such as HyperTerminal of Microsoft Windows:

```
AT+CGMI
```

```
Nokia
```

```
OK
```

### Information Response and Final Result Code

Don't forget the meanings of information response and final result code stated above, since you will see these two terms frequently as you go through this SMS tutorial.

```
AT+CGMI <-- Command line entered
```

```
Nokia <-- Information response
```

```
OK <-- Final result code
```

### Case Sensitivity of AT Commands

In the SMS specification, all AT commands are in uppercase letters. However, many GSM/GPRS modems and mobile phones allow you to type AT commands in either uppercase or lowercase letters. For example, on Nokia 6021, AT commands are case-insensitive and the following two command lines are equivalent:

```
AT+CMGL<CR>
```

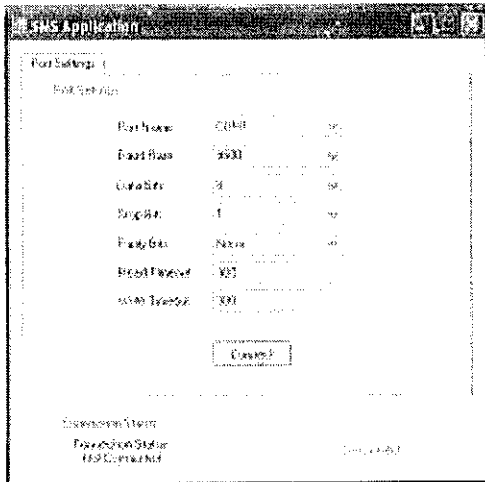
```
at+cmgl<CR>
```



## Sending SMS through GSM Modem using AT Commands

### Port Settings

In this tab, you will have to do port settings which will be the same as you did in hyper terminal and then click the OK button. If modem is connected successfully, a message box will appear with the message “Modem is connected”.



```
public SerialPort OpenPort(string p_strPortName,
    int p_uBaudRate, int p_uDataBits, int p_uReadTimeout, int p_uWriteTimeout)
{
    receiveNow = new AutoResetEvent(false);

    SerialPort port = new SerialPort();

    try
    {
        port.PortName = p_strPortName;           //COM1
```

```

port.BaudRate = p_uBaudRate;           //9600

port.DataBits = p_uDataBits;           //8

port.StopBits = StopBits.One;          //1

port.Parity = Parity.None;             //None

port.ReadTimeout = p_uReadTimeout;     //300

port.WriteTimeout = p_uWriteTimeout;   //300

port.Encoding = Encoding.GetEncoding("iso-8859-1");

port.DataReceived += new SerialDataReceivedEventHandler
                                   (port_DataReceived);

port.Open();

port.DtrEnable = true;

port.RtsEnable = true;
}

catch (Exception ex)

{

    throw ex;

}

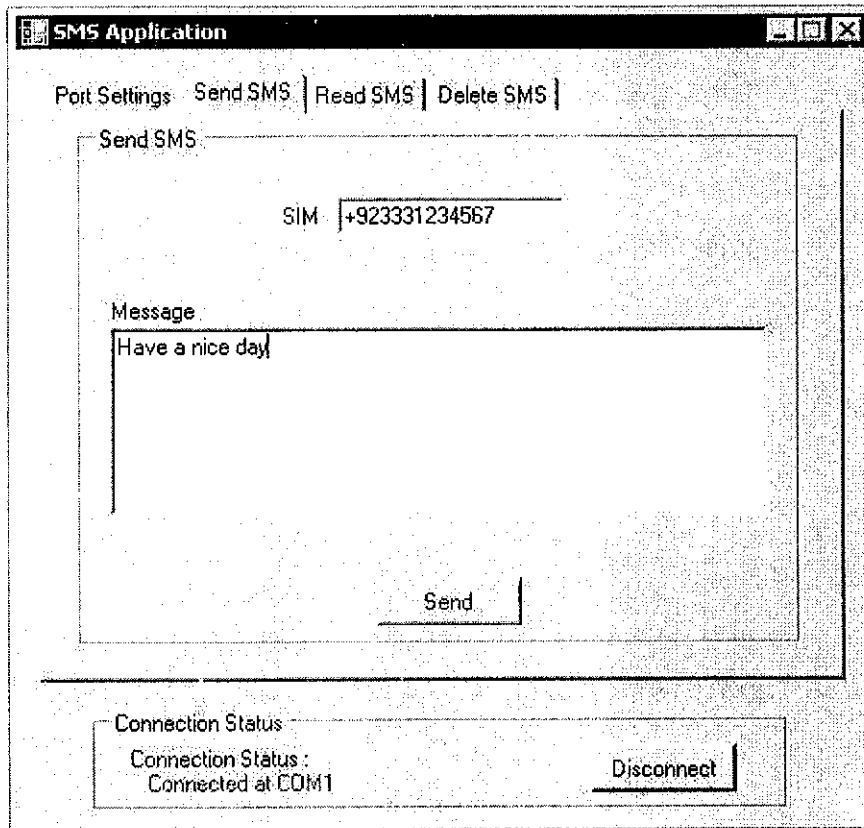
return port;

}

```

## Send SMS

In the second tab, you can send SMS:



```
public bool sendMsg(SerialPort port, string PhoneNo, string Message)
{
    bool isSend = false;

    try
    {
        string recievedData = ExecCommand(port,"AT", 300, "No phone
connected");
```

```

recievedData = ExecCommand(port,"AT+CMGF=1", 300,
                            "Failed to set message format.");

String command = "AT+CMGS=\"" + PhoneNo + "\"";

recievedData = ExecCommand(port,command, 300,
                            "Failed to accept phoneNo");

command = Message + char.ConvertFromUtf32(26) + "\r";

recievedData = ExecCommand(port,command, 3000,
                            "Failed to send message"); //3 seconds

if (recievedData.EndsWith("\r\nOK\r\n"))
{
    isSend = true;
}

else if (recievedData.Contains("ERROR"))
{
    isSend = false;
}

return isSend;
}

catch (Exception ex)
{
    throw new Exception(ex.Message);
}

```

## CHAPTER 8

### TESTING TECHNIQUES IN REAL TIME

#### 8.1. Applying Power for the First Time:

Use a current limited power supply when applying power to the circuit (transmitter and receiver both) board for the first time. By limiting the current, incorrectly inserted components or other parts may survive, but without current limiting, there is a likelihood of permanent damage to the incorrectly inserted components.

#### 8.2. Checking for the Power Supply:

If you observe excessive current, shut off the power supply immediately. If you see zero current, the power may be connected in the wrong polarity, or the IR LED may be installed backwards or not soldered properly. Also, check the voltage regulator connection.

As a final check before proceeding, measure the +5 volt power line on the board (both transmitter and receiver) to verify that the voltage regulator is producing the proper +5 volt power for all the other components on the board.

#### 8.3. Testing the Transmitter Circuit:

The transmitter circuit was tested by applying a voltage of 5 volts to the circuit. Initially, the transmitter circuit was tested by using an ordinary LED in place of an IR LED as viewing of the output of the IR LED is intricate as compared to viewing of the output of an ordinary LED. Then the transmitter circuit was tested by using an IR LED by putting a camera in front of it. If blue rays appeared on the camera screen that implied that the transmitter circuit was transmitting the IR rays correctly.

#### **8.4. Testing the Receiver Circuit:**

The receiver circuit was tested by first switching on the receiver circuit; the alarm rang for a period of 30 seconds for the first time. During the period of 30 seconds, we aligned the transmitter circuit wrt the receiver circuit and after the alignment, if the IR beam was cut or intersected, then the alarm went off again. This correctly demonstrated the proper functioning of both, the transmitter circuit and the receiver circuit.

## CHAPTER 9

### CONTRIBUTION OF THE PROJECT

Our security system provides many features which have not been provided by most expensive systems available in the market. Besides all the processes in our system are automatic in nature, also our system has immense scope for further expansion as we can use multiple IR beams, use face recognition etc. Our system is cheap and made on NE555 timer IC which is very reliable. Also our system is protected by a password making it accessible only to the owner.

Thus the baseline is that we have built a system which has more features than most available systems present, that too at a lower cost.

The areas of application for IRVSMS are:

1. Museums
2. Home security
3. Office security
4. Jeweler shops
5. Banks

## CHAPTER 10

### CONCLUSIONS

The core technology is an integrated security system providing detection of an intrusion, capturing real time video and images of the intrusion and sending sms informing intrusion with a combination of hardware circuits and software codes.

This system is clearly beneficial as it provides utmost security by features like video capturing, image acquisition, backup creation and message sending.

In the last 16 weeks, we have successfully implemented an infrared based surveillance system which provides full proof security.

#### Work performed

- Construction of a full proof security system.
- Use of NE555 timer IC for both the circuits.
- Connection of the webcam with the computer as well as the circuit.
- Code developed for the following:
  - Video capturing,
  - Video saving,
  - Video transferring,
  - Image acquisition,
  - Image saving,
  - Image transferring,
  - SMS sending.



### The problems faced

- Alignment of the IR transmitter and receiver circuit in a straight line.
- Range of IR LED.
- Providing enough power for both the circuit and the webcam.
- Synchronization with computer peripheral.
- Creating a back up of images and videos in real time.
- Soldering of the circuit was time consuming.

## CHAPTER 11

### REFERENCES

- ✦ Yuming Liang and Lihong Xu, "*On Global Path of an IR Beam and Planning for intrusions based on IR Beam Interception*", Shanghai, China, 2009.
- ✦ Tetsuo Asano, David Kirkpatrick and Chee Yap, "*On Pseudo Approximation Algorithms with IR Beam Movement*", Barcelona, Spain, 2002.
- ✦ Wilson D. Esquivel and Luciano E. Chiang, "*IR Beam and IR Mesh Management and requirements for both*" Robotica, Vol.20.No.1, pp49-58, 2002.
- ✦ S. G. KIM and J. H. KIM: *Unmanned IR beams and infrared technology for avoidance of intrusions.*
- ✦ Krishna Nanda Gupta, Prashant Agarwal, Mayur Agarwal: *IR Technology.*
- ✦ James Neufeld, Jason Roberts, Stephen Walsh, Michael Sokolsky, Adam Milstein and Michael Bowling: *Autonomous Geocaching, Navigation and Goal Finding in Outdoor Domains by using infrared technology.*
  
- ✦ [www.ieeexplore.com](http://www.ieeexplore.com)
- ✦ [www.robokits.co.in](http://www.robokits.co.in)
- ✦ [www.alldatasheets.com](http://www.alldatasheets.com)

## CHAPTER 12

### APPENDIX

#### **12.1. MAIN CODE FOR DETECTION OF THE WEBCAM, THE VIDEO STREAMING, IMAGE CAPTURING, IMAGE SAVING, VIDEO SAVING, SMS SENDING AND BACKUP CREATION.**

The following code is the main program code which manages every module.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using System.Net.Sockets;
using System.IO.Ports;

using AForge.Video;
using AForge.Video.DirectShow;
using AForge.Video.VFW;

using InTheHand.Net;
using InTheHand.Net.Bluetooth;
using InTheHand.Net.Sockets;
using InTheHand.Windows.Forms;
```

```

using Brecham.Obex;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private bool DeviceExist = false;
        private FilterInfoCollection videoDevices;
        private VideoCaptureDevice videoSource = null;
        AVIWriter writer = new AVIWriter();
        BluetoothClient client;
        ObexClientSession session;
        Bitmap img,img1;
        Admin obj_form2 = new Admin();
        SerialPort port = new SerialPort();
        clsSMS objclsSMS = new clsSMS();

        public Form1()
        {
            InitializeComponent();
        }

        private void getCamList(object sender, EventArgs e)
        {
            try
            {
                videoDevices = new
                FilterInfoCollection(FilterCategory.VideoInputDevice);
                comboBox1.Items.Clear();
                if (videoDevices.Count == 0)
                    throw new ApplicationException();

                DeviceExist = true;
            }
        }
    }
}

```

```

foreach (FilterInfo device in videoDevices)
{
    comboBox1.Items.Add(device.Name);
}
if(comboBox1.Items.Contains("Webcam-101"))
{
    comboBox1.SelectedIndex = 1;
    start_Click(sender,e);
}
else
{
    getCamList(sender, e);
}

//comboBox1.SelectedIndex = 0; //make default to first cam
}
catch (ApplicationException)
{
    DeviceExist = false;
    comboBox1.Items.Add("No capture device on your system");
}
}

private void rfish_Click(object sender, EventArgs e)
{
    getCamList(sender,e);
}

private void start_Click(object sender, EventArgs e)
{
    if (start.Text == "&Start")
    {

```

```

if (DeviceExist)
{
    videoSource = new
VideoCaptureDevice(videoDevices[comboBox1.SelectedIndex].MonikerString);
    videoSource.NewFrame += new
NewFrameEventHandler(video_NewFrame);
    CloseVideoSource();
    videoSource.DesiredFrameSize = new Size(320,240);
    //videoSource.DesiredFrameRate = 10;
    videoSource.Start();
    label2.Text = "Device running...";
    start.Text = "&Stop";
    timer1.Enabled = true;
    timer2.Enabled = true;
    string datetime = DateTime.Now.ToString("dd-mm-yy hh-mm-ss");
    writer.Open(@"C:\Users\Chammy\Desktop\Surveillance
Video\Video"+datetime+".avi", 320, 240);
    this.port = objclsSMS.OpenPort("com8", 9600, 8, 300, 300);
    if (this.port != null)
    {
        /*if (objclsSMS.sendMsg(this.port, "+919816048232", "Intrusion
Detected at " + DateTime.Now.ToString()))
        {
            MessageBox.Show("Message has been sent successfully");
        }
        else
        {
            MessageBox.Show("Failed to send message");
        }
        */
        this.port.Close();
    }
}
else

```

```

        {
            label2.Text = "Error: No Device selected.";
        }
    }
else
    {
        if (videoSource.IsRunning)
            {
                timer1.Enabled = false;
                timer2.Enabled = false;
                CloseVideoSource();
                label2.Text = "Device stopped.";
                start.Text = "&Start";
                writer.Close();
            }
        }
    }

private void video_NewFrame(object sender, NewFrameEventArgs eventArgs)
{
    img = (Bitmap)eventArgs.Frame.Clone();
    img1 = (Bitmap)eventArgs.Frame.Clone();
    pictureBox1.Image = img;
    writer.FrameRate = 2;
    writer.AddFrame(img1);
}

private void CloseVideoSource()
{
    if (!(videoSource == null))
        if (videoSource.IsRunning)
            {
                videoSource.SignalToStop();
            }
}

```

```

        videoSource = null;
        writer.Close();
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    label2.Text = "Device running... " + videoSource.FramesReceived.ToString()
+ " FPS";
    if (obj_form2.key == 1)
    {
        button4.Text = "Exit Admin";
        comboBox1.Enabled = true;
        comboBox2.Enabled = true;
        rfsh.Enabled = true;
        start.Enabled = true;
        button1.Enabled = true;
        button2.Enabled = true;
        button3.Enabled = true;
        button5.Enabled = true;
        button6.Enabled = true;
    }
    else if (obj_form2.key == 0)
    {
        button4.Text = "Admin";
        comboBox1.Enabled = false;
        comboBox2.Enabled = false;
        rfsh.Enabled = false;
        start.Enabled = false;
        button1.Enabled = false;
        button2.Enabled = false;
        button3.Enabled = false;
        button5.Enabled = false;
    }
}

```



```

        button6.Enabled = false;
    }

}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    CloseVideoSource();
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    CloseVideoSource();
}

private void button1_Click(object sender, EventArgs e)
{
    pictureBox2.Image = pictureBox1.Image;
}

private void button2_Click(object sender, EventArgs e)
{
    string datetime = DateTime.Now.ToString("dd-mm-yy hh-mm-ss");
    pictureBox2.Image.Save(@"C:\Users\Chammy\Desktop\Surveillance
Images\Image"+datetime+".bmp");
}

private void Form1_Load(object sender, EventArgs e)
{
    rfs_Click(sender,e);
    comboBox2.Text = (timer2.Interval/1000).ToString();
}

```

```

private void timer2_Tick(object sender, EventArgs e)
{
    button1_Click(sender, e);
    button2_Click(sender, e);
}

private void button3_Click(object sender, EventArgs e)
{
    Process.Start(@"C:\Users\Chammy\Desktop\Surveillance Images");
}

private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    int timer_changed;
    timer_changed = int.Parse(comboBox2.Text);
    timer_changed = timer_changed * 1000;
    timer2.Interval = timer_changed;
}

private void button5_Click(object sender, EventArgs e)
{
    Process.Start(@"C:\Users\Chammy\Desktop\Surveillance Video");
}

private void button6_Click(object sender, EventArgs e)
{
    using (SelectBluetoothDeviceDialog bldialog = new
SelectBluetoothDeviceDialog())
    {
        bldialog.ShowAuthenticated = true;
        bldialog.ShowRemembered = true;
        bldialog.ShowUnknown = true;
    }
}

```

```

if (bldialog.ShowDialog() == DialogResult.OK)
{
    if (bldialog.SelectedDevice == null)
    {
        MessageBox.Show("No device selected", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    BluetoothDeviceInfo selecteddevice = bldialog.SelectedDevice;
    BluetoothEndPoint remoteEndPoint = new
    BluetoothEndPoint(selecteddevice.DeviceAddress,
        BluetoothService.ObexFileTransfer);

    client = new BluetoothClient();
    try
    {
        client.Connect(remoteEndPoint);
        session = new ObexClientSession(client.GetStream(),
        UInt16.MaxValue);
        session.Connect(ObexConstant.Target.FolderBrowsing);
    }
    catch (SocketException)
    {
        return;
    }
    catch (ObjectDisposedException)
    {
        return;
    }
    catch (IOException)

```

```

        {
            return;
        }
    }
}

if (opdOpenDialog.ShowDialog() == DialogResult.OK)
{
    UploadFiles(opdOpenDialog.FileNames);
}

if (session != null)
{
    try
    {
        session.Disconnect();
        session.Dispose();
    }
    catch { }
}

if (client != null)
{
    client.Close();
    client.Dispose();
}
}

private void UploadFiles(string[] files)
{
    long size = 0;

```

```

List<string> filestoupload = new List<string>();

foreach (string filename in files)
{
    if (File.Exists(filename))
    {
        FileInfo info = new FileInfo(filename);
        filestoupload.Add(filename);
        size += info.Length;
    }
}

using (FileForm upform = new FileForm(new List<string>(filestoupload),
    false, session, size, null))
{
    upform.ShowDialog();
}

private void button4_Click(object sender, EventArgs e)
{
    if (button4.Text == "Admin")
    {
        obj_form2.Show();
    }
    if (button4.Text == "Exit Admin")
    {
        button4.Text = "Admin";
        comboBox1.Enabled = false;
        comboBox2.Enabled = false;
        rfsh.Enabled = false;
        start.Enabled = false;
        button1.Enabled = false;
    }
}

```

```

        button2.Enabled = false;
        button3.Enabled = false;
        button5.Enabled = false;
        button6.Enabled = false;
        obj_form2.key = 0;
    }
}
}
}

```

## **12.2. CODE FOR FILE TRANSFERRING THROUGH BLUETOOTH**

The following code snippet deals with the file transferring wizard it shows the file information and the time remaining, etc. The code shows a new window and a status bar which shows the file transfer progress.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Text;
using System.Windows.Forms;

using Brecham.Obex;

namespace WindowsFormsApplication1
{
    public partial class FileForm : Form

```

```

{
    #region Fields

    public ExceptionDelegate ExceptionMethod;

    ObexClientSession currentSession;
    string dir;
    bool download;
    bool exceptionoccured = false;
    int filesProcessed = 0;
    List<string> filesToProcess;
    long totalsize;

    #endregion Fields

    #region Constructors

    public FileForm(List<string> files, bool down, ObexClientSession
session, long size, string directory)
    {
        InitializeComponent();

        filesToProcess = files;
        download = down;
        currentSession = session;
        totalsize = size;
        dir = directory;
    }

    #endregion Constructors

    #region Public Properties

```

```

public int FilesUploaded
{
    get
    {
        return filesProcessed;
    }
}

#endregion Public Properties

#region Private Methods

private void FileForm_FormClosing(object sender,
FormClosingEventArgs e)
{
    if (bgwWorker.CancellationPending)
    {
        e.Cancel = true;
        return;
    }

    if (bgwWorker.IsBusy)
    {
        if (MessageBox.Show("Cancel operation?", "Confirm",
MessageBoxButtons.OKCancel, MessageBoxIcon.Question) == DialogResult.OK)
        {
            bgwWorker.CancelAsync();
            lblFileCount.Text = "Canceling...";

            while (bgwWorker.IsBusy)
            {
                System.Threading.Thread.Sleep(2000);
            }
        }
    }
}

```



```

        }
        else
            e.Cancel = true;
    }
}

private void FileForm_Shown(object sender, EventArgs e)
{
    if (download)
    {
        lblFileCount.Text = string.Format("Downloading File 1 of
{0}", filesToProcess.Count);
        lblCurrentFile.Text = string.Format("Downloading {0}",
filesToProcess[0]);

        this.Text = "Downloading...";
    }
    else
    {
        lblFileCount.Text = string.Format("Uploading File 1 of {0}",
filesToProcess.Count);
        lblCurrentFile.Text = string.Format("Uploading {0}",
filesToProcess[0]);

        this.Text = "Uploading...";
    }

    bgwWorker.RunWorkerAsync();
}

```

```

private long ProcessStreams(Stream source, Stream destination, long progress,
string filename)

```

```

    {
        byte[] buffer = new byte[1024 * 4];
        while (true)
        {
            //Report downloaded file size
            bgwWorker.ReportProgress(((int)(((progress * 100) /
totalsize)), progress);

            if (bgwWorker.CancellationPending)
            {
                currentSession.Abort();
                return 0;
            }

            try
            {
                int length = source.Read(buffer, 0, buffer.Length);
                if (length == 0) break;
                destination.Write(buffer, 0, length);
                progress += length;
            }
            //Return 0 as if operation was canceled so that processedFiles
is set.

            catch (IOException ex)
            {
                exceptionoccured = true;
                ExceptionMethod(ex);
                return 0;
            }
            catch (ObexResponseException ex)
            {
                exceptionoccured = true;
                ExceptionMethod(ex);

```

```

        return 0;
    }
}
return progress;
}

private void bgwWorker_DoWork(object sender, DoWorkEventArgs e)
{
    long progress = 0;
    DateTime start = DateTime.Now;

    for (int i = 0; i < filesToProcess.Count; i++)
    {
        string currentfile = filesToProcess[i];

        //Report that we started downloading new file
        bgwWorker.ReportProgress((int)((progress * 100) /
totalsize)), i + 1);

        string filename = download ? Path.Combine(dir, currentfile)
: currentfile;

        FileStream hoststream = download ? new
FileStream(filename, FileMode.Create, FileAccess.Write, FileShare.None)
: new FileStream(filename, FileMode.Open,
FileAccess.Read, FileShare.None);

        AbortableStream remotestream = null;
        try
        {
            remotestream = download ?
(AbservableStream)currentSession.Get(currentfile, null)

```

```

:
(AbortableStream)currentSession.Put(Path.GetFileName(currentfile), null);
    }
    catch (IOException ex)
    {
        exceptionoccured = true;
        ExceptionMethod(ex);
        return;
    }
    catch (ObexResponseException ex)
    {
        exceptionoccured = true;
        ExceptionMethod(ex);
        return;
    }
    using (hoststream)
    {
        using (remotestream)
        {
            long result = download ? ProcessStreams(remotestream,
hoststream, progress, currentfile)
: ProcessStreams(hoststream, remotestream, progress,
currentfile);

```

*//Even if we are canceled we need to report how many files we have  
already uploaded*

*//so that they are added to the listview. Or if it is download we need to  
delete the*

*//partially downloaded last file.*

**filesProcessed = i;**

**if (result == 0)**

**{**

```

        e.Cancel = true;
        return;
    }
    else
        progress = result;
    }
}
}
DateTime end = DateTime.Now;
e.Result = end - start;
}

private void bgwWorker_ProgressChanged(object sender,
ProgressChangedEventArgs e)
{
    prbProgress.Value = e.ProgressPercentage;

    if (download)
    {
        if (e.UserState is long)
        {
            lblSize.Text = string.Format("Downloaded {0} of {1}
bytes", (long)e.UserState, totalsize);
        }
        else
        {
            lblFileCount.Text = string.Format("Downloading File {0}
of {1}", (int)e.UserState, filesToProcess.Count);
            lblCurrentFile.Text = string.Format("Downloading {0}",
filesToProcess[(int)e.UserState - 1]);
        }
    }
}
else

```

```

        {
            if (e.UserState is long)
            {
                lblSize.Text = string.Format("Uploaded {0} of {1} bytes",
(long)e.UserState, totalsize);
            }
            else
            {
                lblFileCount.Text = string.Format("Uploading File {0} of
{1}", (int)e.UserState, filesToProcess.Count);
                lblCurrentFile.Text = string.Format("Uploading {0}",
filesToProcess[(int)e.UserState - 1]);
            }
        }
    }

```

```

private void bgwWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
    if (exceptionoccured)
    {
        if (download) File.Delete(Path.Combine(dir,
filesToProcess[filesProcessed]));
        this.Close();
        return;
    }

    if (e.Cancelled)
    {
        if (download) File.Delete(Path.Combine(dir,
filesToProcess[filesProcessed]));

        lblFileCount.Text = "Canceled";
    }
}

```

```

        lblCurrentFile.Visible = false;

        btnCancel.Text = "Canceled";
        this.Text = "Canceled";
    }
    else
    {
        TimeSpan elapsed = (TimeSpan)e.Result;

        lblFileCount.Text = "Done";
        lblCurrentFile.Text = string.Format("Download time: {0}
hours {1} minutes {2} seconds. Average speed: {3} KB/sec",
                                           elapsed.Hours,
                                           elapsed.Minutes,
                                           elapsed.Seconds,
                                           Math.Round(totalsize /
elapsed.TotalSeconds/1024,2));
        prbProgress.Value = 100;
        btnCancel.Text = "Done";
        this.Text = "Done";
    }
}

```

```

private void btnCancel_Click(object sender, EventArgs e)
{
    if (bgwWorker.CancellationPending) return;

    if (bgwWorker.IsBusy)
    {
        bgwWorker.CancelAsync();
        this.Text = lblFileCount.Text = "Canceling...";
    }
}

```

```

        else
            this.Close();
    }

#endregion Private Methods

#region Other

public delegate void ExceptionDelegate(Exception ex);

#endregion Other
}
}

```

### **12.3. SENDING SMS**

The following code is used by the SMS Application. The code snippet is a class which manages opening and closing of communication ports, execution of AT commands, receive and send data from port.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO.Ports;
using System.Threading;
using System.Text.RegularExpressions;

namespace WindowsFormsApplication1

```



```

{
    public class clsSMS
    {

        #region Open and Close Ports
        //Open Port
        public SerialPort OpenPort(string p_strPortName, int p_uBaudRate, int
p_uDataBits, int p_uReadTimeout, int p_uWriteTimeout)
        {
            receiveNow = new AutoResetEvent(false);
            SerialPort port = new SerialPort();

            try
            {
                port.PortName = p_strPortName;           //COM1
                port.BaudRate = p_uBaudRate;           //9600
                port.DataBits = p_uDataBits;           //8
                port.StopBits = StopBits.One;         //1
                port.Parity = Parity.None;            //None
                port.ReadTimeout = p_uReadTimeout;     //300
                port.WriteTimeout = p_uWriteTimeout;   //300
                port.Encoding = Encoding.GetEncoding("iso-8859-1");
                port.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
                port.Open();
                port.DtrEnable = true;
                port.RtsEnable = true;
            }
            catch (Exception)
            {
                //throw ex;
            }
            return port;
        }
    }
}

```

```

    }

    //Close Port
    public void ClosePort(SerialPort port)
    {
        try
        {
            port.Close();
            port.DataReceived -= new
SerialDataReceivedEventHandler(port_DataReceived);
            port = null;
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }

    #endregion

    //Execute AT Command
    public string ExecCommand(SerialPort port,string command, int
responseTimeout, string errorMessage)
    {
        try
        {

            port.DiscardOutBuffer();
            port.DiscardInBuffer();
            receiveNow.Reset();
            port.Write(command + "\n");

            string input = ReadResponse(port, responseTimeout);

```

```

        if ((input.Length == 0) || (!input.EndsWith("\r\n> ") &&
(!input.EndsWith("\r\nOK\r\n"))))
            throw new ApplicationException("No success message was received.");
        return input;
    }
    catch (Exception ex)
    {
        throw new ApplicationException(errorMessage, ex);
    }
}

```

//Receive data from port

```

public void port_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    try
    {
        if (e.EventType == SerialData.Chars)
            receiveNow.Set();
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

```

```

public string ReadResponse(SerialPort port,int timeout)

```

```

{
    string buffer = string.Empty;
    try
    {
        do
        {
            if (receiveNow.WaitOne(timeout, false))
            {

```

```

        string t = port.ReadExisting();
        buffer += t;
    }
    else
    {
        if (buffer.Length > 0)
            throw new ApplicationException("Response received is
incomplete.");
        else
            throw new ApplicationException("No data received from phone.");
    }
}
while (!buffer.EndsWith("\r\nOK\r\n") && !buffer.EndsWith("\r\n> ") &&
!buffer.EndsWith("\r\nERROR\r\n"));
}
catch (Exception ex)
{
    throw ex;
}
return buffer;
}

```

#region Read SMS

```
public AutoResetEvent receiveNow;
```

```
public ShortMessageCollection ParseMessages(string input)
```

```
{
    ShortMessageCollection messages = new ShortMessageCollection();
    try
    {

```

```

        Regex r = new Regex(@"^+CMGL:
(\d+);""(+)"";""(+)"".(.*)""(+)""\n(+)r\n");
        Match m = r.Match(input);
        while (m.Success)
        {
            ShortMessage msg = new ShortMessage();
            //msg.Index = int.Parse(m.Groups[1].Value);
            msg.Index = m.Groups[1].Value;
            msg.Status = m.Groups[2].Value;
            msg.Sender = m.Groups[3].Value;
            msg.Alphabet = m.Groups[4].Value;
            msg.Sent = m.Groups[5].Value;
            msg.Message = m.Groups[6].Value;
            messages.Add(msg);

            m = m.NextMatch();
        }

    }
    catch (Exception ex)
    {
        throw ex;
    }
    return messages;
}

#endregion

#region Send SMS

static AutoResetEvent readNow = new AutoResetEvent(false);

public bool sendMsg(SerialPort port, string PhoneNo, string Message)

```

```

    {
        bool isSend = false;

        try
        {

            string recievedData = ExecCommand(port,"AT", 300, "No phone
connected");
            recievedData = ExecCommand(port,"AT+CMGF=1", 300, "Failed to set
message format.");
            String command = "AT+CMGS=\"" + PhoneNo + "\"";
            recievedData = ExecCommand(port,command, 300, "Failed to accept
phoneNo");
            command = Message + char.ConvertFromUtf32(26) + "\r";
            recievedData = ExecCommand(port,command, 5000, "Failed to send
message"); //3 seconds
            if (recievedData.EndsWith("\r\nOK\r\n"))
            {
                isSend = true;
            }
            else if (recievedData.Contains("ERROR"))
            {
                isSend = false;
            }
            return isSend;
        }
        catch (Exception)
        {
            return isSend;
            //throw new Exception(ex.Message);
        }
    }

```

```

static void DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    try
    {
        if (e.EventType == SerialData.Chars)
            readNow.Set();
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

#endregion

}
}

```

#### **12.4. BACKEND CODE FOR VARIOUS COMPONENTS-BUTTONS, MENU, VIDEO SPACE, ETC.**

The backend code developed by the visual studio tool which has information about the main form. The various components, buttons, menu, video space, etc.

```

namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>

```

```

private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed:
otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.pictureBox1 = new System.Windows.Forms.PictureBox();
    this.groupBox1 = new System.Windows.Forms.GroupBox();
    this.label2 = new System.Windows.Forms.Label();
    this.start = new System.Windows.Forms.Button();
    this.rfsh = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
}

```



```

this.comboBox1 = new System.Windows.Forms.ComboBox();
this.timer1 = new System.Windows.Forms.Timer(this.components);
this.pictureBox2 = new System.Windows.Forms.PictureBox();
this.button1 = new System.Windows.Forms.Button();
this.button2 = new System.Windows.Forms.Button();
this.timer2 = new System.Windows.Forms.Timer(this.components);
this.button3 = new System.Windows.Forms.Button();
this.label3 = new System.Windows.Forms.Label();
this.label4 = new System.Windows.Forms.Label();
this.comboBox2 = new System.Windows.Forms.ComboBox();
this.button4 = new System.Windows.Forms.Button();
this.button5 = new System.Windows.Forms.Button();
this.button6 = new System.Windows.Forms.Button();
this.opdOpenDialog = new System.Windows.Forms.OpenFileDialog();
((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
this.groupBox1.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.pictureBox2)).BeginInit();
this.SuspendLayout();
//
// pictureBox1
//
this.pictureBox1.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
this.pictureBox1.Location = new System.Drawing.Point(338, 10);
this.pictureBox1.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.pictureBox1.Name = "pictureBox1";
this.pictureBox1.Size = new System.Drawing.Size(319, 240);
this.pictureBox1.TabIndex = 0;
this.pictureBox1.TabStop = false;
//
// groupBox1
//
this.groupBox1.Controls.Add(this.label2);

```

```

this.groupBox1.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.groupBox1.Location = new System.Drawing.Point(72, 150);
this.groupBox1.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Padding = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.groupBox1.Size = new System.Drawing.Size(181, 32);
this.groupBox1.TabIndex = 11;
this.groupBox1.TabStop = false;
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(2, 13);
this.label2.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(83, 15);
this.label2.TabIndex = 0;
this.label2.Text = "Device ready.";
//
// start
//
this.start.Enabled = false;
this.start.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.start.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
(byte)0));
this.start.Location = new System.Drawing.Point(187, 111);
this.start.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.start.Name = "start";
this.start.Size = new System.Drawing.Size(61, 25);
this.start.TabIndex = 10;
this.start.Text = "&Start";
this.start.UseVisualStyleBackColor = true;

```

```

this.start.Click += new System.EventHandler(this.start_Click);
//
// rfish
//
this.rfsh.Enabled = false;
this.rfsh.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.rfsh.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));
this.rfsh.Location = new System.Drawing.Point(85, 111);
this.rfsh.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.rfsh.Name = "rfsh";
this.rfsh.Size = new System.Drawing.Size(61, 25);
this.rfsh.TabIndex = 9;
this.rfsh.Text = "&Refresh";
this.rfsh.UseVisualStyleBackColor = true;
this.rfsh.Click += new System.EventHandler(this.rfsh_Click);
//
// label1
//
this.label1.AutoSize = true;
this.label1.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));
this.label1.Location = new System.Drawing.Point(69, 35);
this.label1.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(113, 15);
this.label1.TabIndex = 8;
this.label1.Text = "Select video source";
//
// comboBox1

```

```

//
this.comboBox1.Enabled = false;
this.comboBox1.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.comboBox1.FormattingEnabled = true;
this.comboBox1.Location = new System.Drawing.Point(71, 67);
this.comboBox1.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.comboBox1.Name = "comboBox1";
this.comboBox1.Size = new System.Drawing.Size(201, 23);
this.comboBox1.TabIndex = 7;
//
// timer1
//
this.timer1.Interval = 1000;
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
//
// pictureBox2
//
this.pictureBox2.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
this.pictureBox2.Location = new System.Drawing.Point(12, 224);
this.pictureBox2.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.pictureBox2.Name = "pictureBox2";
this.pictureBox2.Size = new System.Drawing.Size(319, 240);
this.pictureBox2.TabIndex = 12;
this.pictureBox2.TabStop = false;
//
// button1
//
this.button1.Enabled = false;
this.button1.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button1.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));

```

```

this.button1.Location = new System.Drawing.Point(338, 290);
this.button1.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(65, 25);
this.button1.TabIndex = 13;
this.button1.Text = "&Capture";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// button2
//
this.button2.Enabled = false;
this.button2.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button2.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));
this.button2.Location = new System.Drawing.Point(338, 385);
this.button2.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(65, 25);
this.button2.TabIndex = 14;
this.button2.Text = "S&ave";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);
//
// timer2
//
this.timer2.Interval = 3000;
this.timer2.Tick += new System.EventHandler(this.timer2_Tick);
//
// button3
//
this.button3.Enabled = false;

```

```
this.button3.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button3.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));

this.button3.Location = new System.Drawing.Point(443, 301);
this.button3.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(83, 38);
this.button3.TabIndex = 15;
this.button3.Text = "&Open Image Folder";
this.button3.UseVisualStyleBackColor = true;
this.button3.Click += new System.EventHandler(this.button3_Click);
//
// label3
//
this.label3.AutoSize = true;
this.label3.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.label3.Location = new System.Drawing.Point(464, 369);
this.label3.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(151, 15);
this.label3.TabIndex = 16;
this.label3.Text = "Set Image Capture Interval";
//
// label4
//
this.label4.AutoSize = true;
this.label4.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.label4.Location = new System.Drawing.Point(553, 391);
this.label4.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(32, 15);
this.label4.TabIndex = 18;
```

```

this.label4.Text = "secs";
//
// comboBox2
//
this.comboBox2.AllowDrop = true;
this.comboBox2.Enabled = false;
this.comboBox2.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.comboBox2.FormattingEnabled = true;
this.comboBox2.Items.AddRange(new object[] {
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9"});
this.comboBox2.Location = new System.Drawing.Point(513, 388);
this.comboBox2.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.comboBox2.Name = "comboBox2";
this.comboBox2.Size = new System.Drawing.Size(31, 23);
this.comboBox2.TabIndex = 19;
this.comboBox2.SelectedIndexChanged += new
System.EventHandler(this.comboBox2_SelectedIndexChanged);
//
// button4
//
this.button4.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button4.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));
this.button4.Location = new System.Drawing.Point(603, 430);

```

```

this.button4.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button4.Name = "button4";
this.button4.Size = new System.Drawing.Size(56, 32);
this.button4.TabIndex = 20;
this.button4.Text = "A&dmin";
this.button4.UseVisualStyleBackColor = true;
this.button4.Click += new System.EventHandler(this.button4_Click);
//
// button5
//
this.button5.Enabled = false;
this.button5.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button5.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
(byte)0));
this.button5.Location = new System.Drawing.Point(546, 301);
this.button5.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button5.Name = "button5";
this.button5.Size = new System.Drawing.Size(87, 38);
this.button5.TabIndex = 21;
this.button5.Text = "O&pen Video Folder";
this.button5.UseVisualStyleBackColor = true;
this.button5.Click += new System.EventHandler(this.button5_Click);
//
// button6
//
this.button6.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.button6.Location = new System.Drawing.Point(443, 434);
this.button6.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
this.button6.Name = "button6";
this.button6.Size = new System.Drawing.Size(75, 29);
this.button6.TabIndex = 22;
this.button6.Text = "&Backup";

```



```

this.button6.UseVisualStyleBackColor = true;
this.button6.Click += new System.EventHandler(this.button6_Click);
//
// opdOpenDialog
//
this.opdOpenDialog.Multiselect = true;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.Size(7F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.SystemColors.ButtonHighlight;
this.BackgroundImageLayout =
System.Windows.Forms.ImageLayout.Stretch;
this.ClientSize = new System.Drawing.Size(667, 475);
this.Controls.Add(this.button6);
this.Controls.Add(this.button5);
this.Controls.Add(this.button4);
this.Controls.Add(this.comboBox2);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.button3);
this.Controls.Add(this.button2);
this.Controls.Add(this.button1);
this.Controls.Add(this.pictureBox2);
this.Controls.Add(this.groupBox1);
this.Controls.Add(this.start);
this.Controls.Add(this.rfsh);
this.Controls.Add(this.label1);
this.Controls.Add(this.comboBox1);
this.Controls.Add(this.pictureBox1);

```

```

        this.Font = new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)0));
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
        this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
        this.Margin = new System.Windows.Forms.Padding(2, 5, 2, 5);
        this.MaximizeBox = false;
        this.MinimizeBox = false;
        this.Name = "Form1";
        this.Text = "Video Surveillance";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.FormClosed += new
System.Windows.Forms.FormClosedEventHandler(this.Form1_FormClosed);
        ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
        this.groupBox1.ResumeLayout(false);
        this.groupBox1.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBox2)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

```

#endregion

```

private System.Windows.Forms.PictureBox pictureBox1;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Button start;
private System.Windows.Forms.Button rfs;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.ComboBox comboBox1;
private System.Windows.Forms.Timer timer1;
private System.Windows.Forms.PictureBox pictureBox2;

```

```

private System.Windows.Forms.Button button1;
private System.Windows.Forms.Button button2;
private System.Windows.Forms.Timer timer2;
private System.Windows.Forms.Button button3;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.ComboBox comboBox2;
private System.Windows.Forms.Button button4;
private System.Windows.Forms.Button button5;
private System.Windows.Forms.Button button6;
private System.Windows.Forms.OpenFileDialog opdOpenDialog;
}
}

```

## **12.5. CODE FOR PASSWORD AUTHENTICATION**

The authentication code snippet. The program authenticates the user and checks for the correct password, if the password is correct the user gets through and gets additional privileges.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Admin : Form

```

```

{
    public int key = 0;
    private string passwd = "temp";

    public Admin()
    {
        InitializeComponent();
    }

    private void textBox1_KeyUp(object sender, KeyEventArgs e)
    {
        try
        {
            if (e.KeyCode == Keys.Enter)
            {
                button1_Click(sender, e);
            }
            if (e.KeyCode == Keys.Escape)
            {
                Hide();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        if (textBox1.Text == passwd)
        {

```

```

        key = 1;
        textBox1.Text = "";
        Hide();
    }
else
    {
        textBox1.DeselectAll();
        textBox1.Text = "";
        MessageBox.Show("Incorrect Password");
        textBox1.Text = "";
        textBox1.Select();
    }
}
}
}
}
}

```

## **12.6. BACKEND CODE FOR VARIOUS BUTTONS, TEXT BOXES, ETC.**

The following code snippet is the visual studio generated code. It deals with the various buttons and the texts box, etc.

```

namespace WindowsFormsApplication1
{
    partial class Admin
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

```

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed:
otherwise, false.</param>
```

```
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

```
#region Windows Form Designer generated code
```

```
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // textBox1
    //
    this.textBox1.AcceptsReturn = true;
    this.textBox1.Location = new System.Drawing.Point(5, 8);
    this.textBox1.Name = "textBox1";
    this.textBox1.PasswordChar = '*';
```

```

        this.textBox1.Size = new System.Drawing.Size(127, 20);
        this.textBox1.TabIndex = 0;
        this.textBox1.KeyUp += new
System.Windows.Forms.KeyEventHandler(this.textBox1_KeyUp);
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(135, 8);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(39, 20);
        this.button1.TabIndex = 1;
        this.button1.Text = "OK";
        this.button1.UseVisualStyleBackColor = true;
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // Admin
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.BackColor = System.Drawing.SystemColors.HighlightText;
        this.ClientSize = new System.Drawing.Size(178, 36);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.textBox1);
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
        this.MaximizeBox = false;
        this.MinimizeBox = false;
        this.Name = "Admin";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "Admin";
        this.ResumeLayout(false);
        this.PerformLayout();
    }

```

```
#endregion
```

```
private System.Windows.Forms.TextBox textBox1;
```

```
private System.Windows.Forms.Button button1;
```

```
}
```

```
}
```