DEVELOPMENT OF THE WIRELESS INSTRUCTOR SYSTEM AND BLUETOOTH HANDOVER TECHNOLOGIES FOR IMPROVED VIRTUAL LABORATORY APPLICATIONS

By

LYNN MOSES GEORGE

Master of Science

Oklahoma State University

Stillwater, Oklahoma

2003

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August, 2003

DEVELOPMENT OF THE WIRELESS INSTRUCTOR SYSTEM AND BLUETOOTH HANDOVER TECHNOLOGIES FOR IMPROVED VIRTUAL LABORATORY APPLICATIONS

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

Timothy J. Pettibone

Dean of the Graduate College

PREFACE

The thesis presents the details of features and implementations of the Virtual Lab developed in the Telecommunication Laboratory (104 Gundersen Hall) of Oklahoma State University using Bluetooth as the crucial wireless link between the Instructor and the students. In a laboratory course where the lab is comprised of many rooms, the essential interactivity between students is lost as the Instructor needs to move between rooms. The Instructor does not have the capability to lecture and advice the students as the exercises are being done. The lecture session and the exercise session need to be separate. Moreover, distant students definitely need to travel to the campus to attend the Lab courses. These issues pertaining to a laboratory course were addressed and a solution, that took the form, Virtual Lab, has been formulated and developed. The project has been funded by the Department of Education for bringing up the interaction and application of high end technology in education. The Instructor is given the capability of moving around the rooms of a lab and always have contact with the students through the Compaq iPAQ PocketPC, which has been enhanced to function as per the requirements for the virtual lab through an application program developed for the purpose. The instructor is capable of sending files to the students from any place in the lab and the students are always aware of the Instructor's location and could page the Instructor with text messages. Moreover the students have been provided the added advantage of being

able to see what the Instructor sees and listen to the lecture no matter which room the Instructor is in.

The movement of the Instructor requires the wireless links to be checked for quality as the Instructor moves from one room to another, tear down bad links and form new stronger links. This procedure named Handover needs to be done in a manner that the normal traffic is not affected. The virtual lab needed efficient and fast algorithms for Handover and hence three new algorithms were developed and implemented and analyzed for speed. The virtual lab has been equipped with a Handover mechanism for switching for maintaining a strong link wherever the Instructor may move in the Lab.

ACKNOWLEDGEMENT

This is my moment to thank the Almighty Lord for blessing me with more than what I am worth and placing me in the right places and with the right people all my life.

I would like to take this opportunity to thank my advisor, Dr. Jong-Moon Chung for his constant guidance, support and insight that he has offered devoid of which growth would have been impossible. I would also like to thank my fellow members, Lijy Kallidukil for her role with the ideas of Bluetooth Handover, Yuan Zhang for developing the Instructor Access Management utility and Theepa Sudha Natarajan for developing the Configuration Utility and everyone in the Advanced Communication Systems Engineering Laboratory (ACSEL) and Oklahoma Communication Laboratory for Networking and Bioengineering (OCLNB) who have made these 2 years a pleasant experience.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter I

Introduction

Telecommunication brought the world closer and wireless communication is poised to make the world move freely and it's well on the way to achieve it. In the future, users will no longer have to constrain themselves to computers and telephones connected to a hub on the wall. From anywhere and at anytime, communication will be possible. And wireless technologies that are strong in distance, data rate and security are being built with a mind to the application area. Cellular technologies service subscribers with voice and data communication over distances in kilometers but with very low data rates. Wireless Local Area Networks (WLAN) serve users with high data rate communication over distances in the range of meters. And then there are the Wireless Personal Area Networks (WPAN) that are of short range and are designed for low power consumption, low price and for accommodating more users in a small area.

Wireless technology could be applied to the education system to enhance the learning experience through application of the technology in areas that where interactivity and closeness with the students are inhibited. One such application would be in a Lab course in a University. A lab course requires greater interactivity of the students with the instructor, but a lab with many small rooms may reduce the interactivity to a level lesser than that in a theory class. Moreover distant students are forced to travel to the campus in order to attend lab courses. All these problems have been addressed and a solution that is named the Virtual Lab has been developed, which is funded by the

Department of Education (DOE). The Instructor is connected with the students wherever he/she may be in the lab through the Compaq IPAQ PocketPC, that has been enhanced to integrate as the main part of the Virtual Lab. This has been done by extensive programming to create an application program that would run on the PocketPC. The Instructor is now capable of sending files to the students and receiving instant text messages from them regardless of his/her location in the lab. The location of the Instructor is made known to the students as well and this information is constantly refreshed. All these tasks that are accomplished by the PocketPC are made possible through the wireless link it makes with the wired network. The wireless technology used here is Bluetooth and the PocketPC is a part of a WPAN. The Instructor is also equipped with a wireless audio/video headset that would allow the students to listen to the instructor wherever he/she may be and see exactly what the instructor sees. This could help the Instructor in explaining exercises in a better manner.

As the Instructor moves, so should the wireless link, that is, a weaker link should be torn down and a new stronger link needs to be created in its place with a nearby Access point. This procedure is called Handover and the Virtual Lab needs a Handover algorithm so that the Instructor would always be provided with a good quality wireless link always. The Handover procedure itself should cause only minimum disturbance to the normal traffic flow in the wireless link. Considering these facts, three new algorithms for Bluetooth handover was developed and analyzed. The Virtual Lab is implemented with a basic algorithm for Bluetooth handover.

The thesis provides details of the Virtual lab and the new algorithms for Bluetooth Handover that can also aid the Virtual Lab application.

1.1 Wireless Personal Area Networks

Wireless Personal Area Networks are aimed at small mobile devices; mainly Personal Digital Assistants (PDA), pagers, cell phones, and laptops which personalizes the wireless connection to the network with the user's information automatically or by the user invoking it. The user may be uniquely identified by the connection and personalized data could be downloaded. The main advantage of this type of wireless network is the feature of the device automatically connecting to the network and downloading or uploading personalized information, in the so called Personal Operating Space (POS) surrounding the user which is usually a range of 10 meters or less. The user may be able to connect to the outside world as well through the network but it may offer a very low data rate.

1.2 Bluetooth

Bluetooth is a robust, low cost, low power short-range radio technology that operates in the unlicensed Industrial, Scientific, and Medical (ISM) band at 2.4 GHz. The Bluetooth system provides point-to-point connection or point-to-multipoint connection over a scatternet, in which a frequency hopping channel is shared among several units through the time division duplex (TDD) multiplexing topology. Bluetooth systems apply a versatile short-range WPAN technology that supports various rates of voice and data communications. The specifications for Bluetooth were also adopted as the IEEE 802.15 standards. Such a network will have Base Stations that will connect to the Bluetooth devices and contribute to the flow of data between the devices and to the outside world.

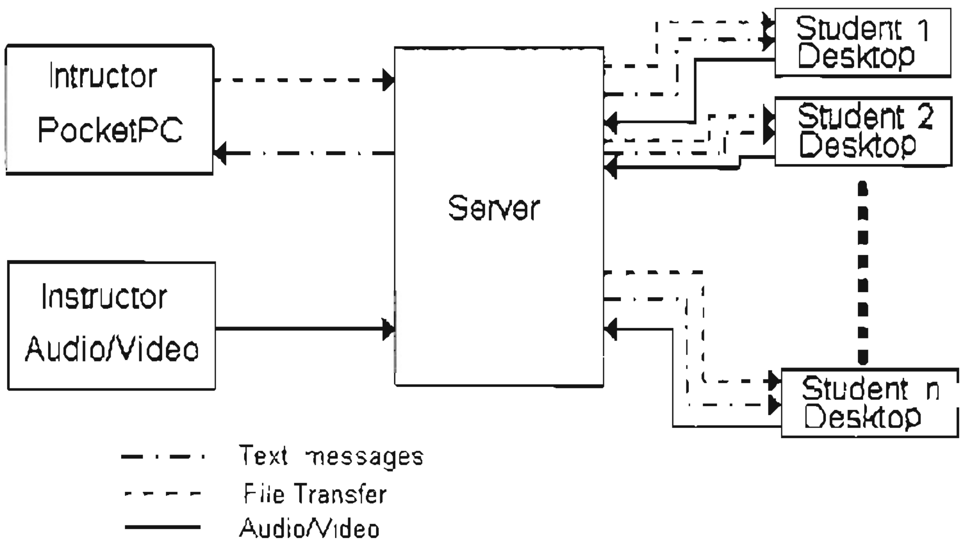


Fig. 1.1. Virtual Lab setup.

1.4 Need for a Virtual Lab

Instructors handling lab courses where the labs have many rooms have many challenges in lecturing and carrying on with the exercises. Many work-around solutions are used by the instructors compromising on interactivity and time. The problems could be fixed if technology could be put to use the right way.

1.4.1 Mobility of Instructor

In a Lab where there are many rooms, the Instructor may be constrained to instruct students in each room separately. The instructor may have to repeat the same instructions over again in each room or the students may have to gather in a bigger room for a lecture session before the practical session. It is difficult for the lecture and the exercises to be carried on simultaneously. This scenario could be avoided by giving mobility to the Instructor so that the students can listen to the lecture even as the instructor moves across the rooms.

1.4.2 Interactivity for Remote Students

For lab courses, remote students need to travel to the location of the lab to attend classes. Though theoretical classes are offered for Remote students audio-visually, lab classes have been constrained by the movement of the instructor and the need for observation of the instruments and exercises in closer detail by the students. Moreover, the interactivity of the remote students is lost when the Instructor has to move around the rooms of the lab.

1.4.3 Ease of accessibility of Information for Instructor

The instructor has to be moving around the rooms of a lab to instruct the students and when the students have to be distributed with the manuals or some important files need to be transferred to the students, the Instructor has to go to a desktop computer. The instructor may also at times need to get information from the Internet for various purposes. In this case too, the instructor needs to go back to a desktop computer. This problem needs to be solved so that the instructor could be able to access the Internet as well as distribute information to the students from his/her location.

1.5 Bluetooth Handover

Handover in a wireless communication is the flow of procedures that leads a mobile device to detach from a central control system and connect to another due to the growing distance with the former and the proximity with the latter. There are two types of handovers – Hard and Soft. In a Hard Handover, the old link is broken before a new link is formed and hence there is an interruption in the traffic while in Soft Handover, a new link is first created before the old one is brought down and hence the flow of traffic is almost undisturbed. Bluetooth is a small range radio network and hence handover becomes a very important part when it is used in large areas. This report presents 3 new algorithms for Hard as well as Soft Handover in Bluetooth. The general procedure of Handover is depicted in Fig.1.2. When a certain device having a link with an Access Point (AP) moves away and closer to another AP it detaches the old link and forms a new one with the nearby AP.

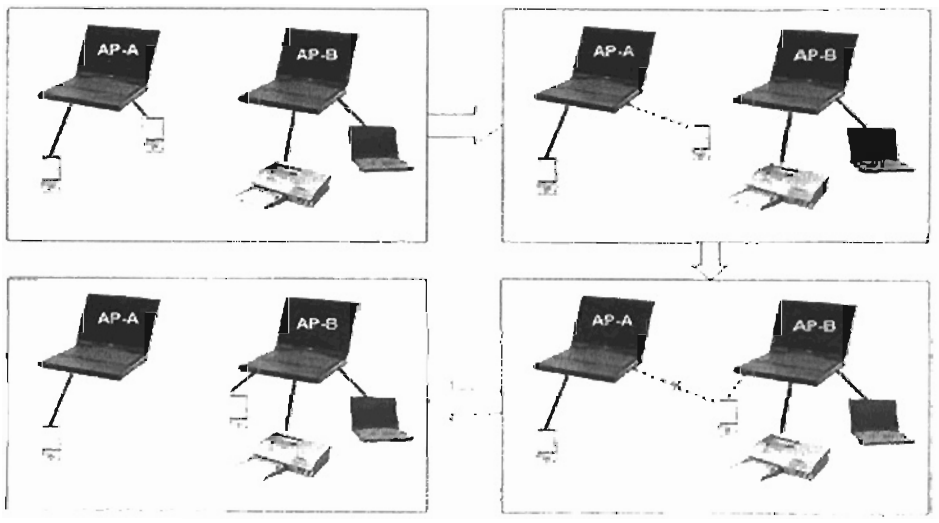


Fig. 1.2 General procedure of Handover

1.6 Overview of existing Handover technologies and the improvements made in this thesis

Bluetooth handover is made possible by measuring the received signal strength indicator (RSSI) value of the link. In [5] and [6], the authors propose a handover technique where the search for the new device to connect to is initiated once the link signal quality goes down below a predefined threshold. The potential problem for this approach of searching for new devices when the signal quality goes down is that the search process requires a lot of time (in the order of seconds) which consequently elongates the Handover delay. The handover methodologies proposed in this thesis reduce the handover timing by removing this process out during handover. In [6] and [7], the authors discuss about hard handover approaches, in which the technique accomplishes handover in the Internet protocol (IP) layer, which means that the handover process is based on a connectionless path. The mobile handset looks for the IP address of the strongest signal in the neighborhood and tries to connect to it. The hard handoff procedures are supported by updating the adjacent Bluetooth routing tables. The papers also propose a technique where when a certain mobile device, say A, hands-over to a new Base Station, another mobile device already a slave in the new Base station would act as a repeater and serve the mobile handset, A till the Handover is complete. This has the disadvantage that any mobile device has to keep track of not only the nearby Base stations but also the mobile devices associated with them to connect to during handover and the serving mobile device has to be an active participant in two piconets during the period. In [8], the re-routing in layer-3 during handover is explained, which could be used regardless of the method of Bluetooth handover used in layer-2. Our paper proposes three methodologies to do Handover in Layer-2 which uses Bluetooth.

1.7 Thesis Outline

Chapter I has given a basic introduction and idea of what to expect from the rest of the thesis. Chapter II will present a Literature review of Bluetooth and the needs of the Virtual Lab. Chapter III presents the details of the features and implementations of the

Virtual Lab. Chapter IV presents the three new Bluetooth Handover Algorithms along with the analysis of the timing for each algorithm. Chapter V sums up the work done in each topic and the conclusions drawn from the analysis of the developments.

Chapter II

Literature Review and Overview of Existing Technologies

This chapter gives a brief review of the architecture of Bluetooth and its physical layer properties. This chapter also discusses the other possible ways in which the Virtual Lab could have been implemented and further touches on why the ideas were dropped.

2. 1 Description of Bluetooth

Bluetooth is a short range radio technology aimed at Personal Area Networking. Each Bluetooth device could communicate with each other only after forming a wireless link with the device.

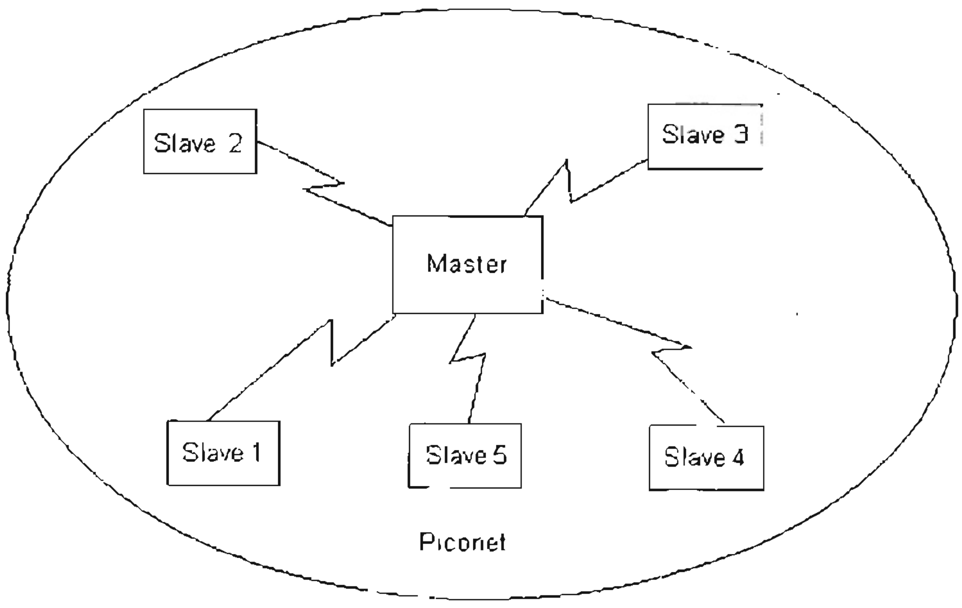


Fig 2 : Master-Slave Architecture of Bluetooth

In Bluetooth, the devices need not have a direct connection with each other for being able to communicate. Each device is linked or associated with a Master which is able to control 7 other devices which are called Slaves as shown in Fig. 2.1. The traffic flow between the devices is controlled and regulated by the Master. Bluetooth devices are classified into 3 classes namely Class 1, Class 2 and Class 3. These classifications are based on the maximum power of transmission and the power levels are 100 mW, 2.5 mW and 1 mW for Class 1, Class 2 and Class 3 respectively.

2.2 Physical Layer Specifications of Bluetooth

Bluetooth is a Frequency Hopping Spread Spectrum technology in the physical layer. This means the data is sent at a particular frequency for a certain period and then switches to a different frequency, which can be determined by different means. Bluetooth hops around the ISM band of 2.4 GHz to 2.84 GHz 1600 times every second, which means the frequency of transmission is changed 1600 times every second as shown in Fig.2.2. Each hop or transmission is of a bandwidth of 1 MHz. The Frequency hopping sequence is generated by using the Bluetooth Address and Bluetooth clock of a device. Bluetooth devices or slaves communicate with each other via the Master. The master and slaves communicate with each other by TDD method. The master sends a "Poll" packet to each slave and the slave responds with a data packet in response to it or a "Null" packet if it has no data to send. Each time slot is of 625 microseconds width and the frequency of transmission is same for the whole slot. The Master always sends its packet in the even slot and the slave responds in the subsequent slot which is odd. The master and all its slaves form an entity termed as the piconet in Bluetooth. All devices in a piconet follow the Frequency Hopping sequence generated using the Master's Bluetooth address and clock. In this way all the devices in a piconet are synchronized.

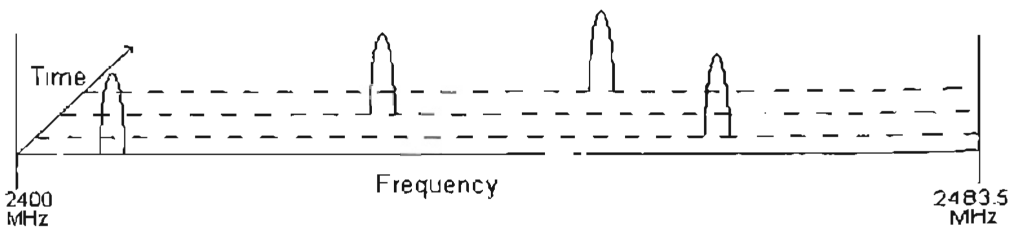


Fig 2.2 Physical Layer Specification of Bluetooth

2.3 Scotty FTP API

The Virtual Lab needed lot of file transfers ~~to be~~ done from the PocketPC to the server. This was decided to be done using the File Transfer Protocol (FTP). The Wireless Instructor System program on the PocketPC was programmed using the Application Program Interface (API) provided by Microsoft for Windows CE. But the FTP functions did not work and later it was found that the API's had bugs that had not been fixed by Microsoft yet. So, being in need of API's to implement the FTP functions, the internet was searched and one Scotty FTP API was found that was provided by a company in India named Ruksun Software. After talks with the company, the API was donated to Oklahoma State University by Ruksun. The API has been used in the Wireless Instructor System developed on the PocketPC and is working very well.

2.4 Some critical decisions for Virtual Lab

The Virtual Lab could have been designed in a different way than it had been designed now. Various alternatives were weighed upon and the best decisions that suited the cost and gave the best performance were chosen.

2.4.1 Why Bluetooth?

Bluetooth is a Personal Area Network technology which itself fits into the project very well as the wireless technology to be used in the Virtual project needs to attach itself to the network on its own, configure and personalize itself as well as download relevant information. Bluetooth also allows the user to browse internet and transfer files through FTP. Bluetooth is cheaper than IEEE 802.11b. Though Bluetooth has a smaller data rate than IEEE 802.11b considering cost and actual data rate needed by the instructor, Bluetooth fits the role well.

2.4.2 Reasons for rejecting Infrared from being used for Instructor location

The location of the Instructor in the lab could be traced by using many means. One of the most considered was Infrared (IR). It was proposed that an IR receiver be placed on the wall opposite every door and the PocketPC could be sending beams of IR rays periodically. The IR rays have a particular code for the instructor who is carrying the PocketPC. When the Instructor enters the room, the IR from the PocketPC could be detected by the IR receiver which then sends electrical signals, according to the code in the IR ray, to a computer to which it has been wired. This information could be collected in a server and thus the instructors could be located at any time. The advantage of IR is that it is highly directional and hence it would not go out of a room and hence there is no possibility of locating the instructor wrongly. But, the directionality itself is the problem too. Since IR rays are highly directional there is a possibility that the IR rays may miss the receiver quite often. Not only this, the wiring to the computers from the receiver may prove strenuous to maintain. So this idea was dropped and Bluetooth itself was manipulated to be used for Instructor location.

2.4.3 Audio Video Headset compatibility with Bluetooth

The audio/video Headset plays a major role in the interactivity that the Virtual Lab provides to the participants. Bluetooth functions in the 2.4 GHz ISM band. This band is a free band for developers and has minimum regulations. Most of the audio/video transmitters were found to function in the same band. This is would pose a problem to the reliability of the Bluetooth link as well as the quality of the video transfer. So, lot of audio/video transmitters were found and analyzed and the best components that were both low in cost, good in performance and that avoided the 2.4 GHz ISM band were chosen and were implemented. The components were purchased from the vendor, Microtek, Inc. and the Transmitter/Receiver function at the ISM 900MHz band.

Chapter III

Features and the Implementation Specifics of the Virtual Lab

Virtual Lab is implemented by using Bluetooth along with the wired Ethernet network. The Compaq iPAQ PocketPC is used for data transfer between the instructor and the students and the Audio/Video device is used for transmission of Audio and Video to the students. An intermediate server acts the part of organizing and distributing the data between the instructor and the students.

3.1 Features of the Virtual Lab

Web Pushing:

Web Pushing is a feature on the PocketPC to transfer files to the server from where the students could pick them. This has been implemented using FTP.

Instructor location:

Instructor location is a feature that would help students to locate the position of the instructor in the lab. This would also help them in controlling the camera in the appropriate room so as to view the Instructor. The virtual lab is implemented as depicted in Fig 3.1. and the information of the LAN Access Point to which the PocketPC is attached to is passed to the server, and thereby the location of the Instructor is known.

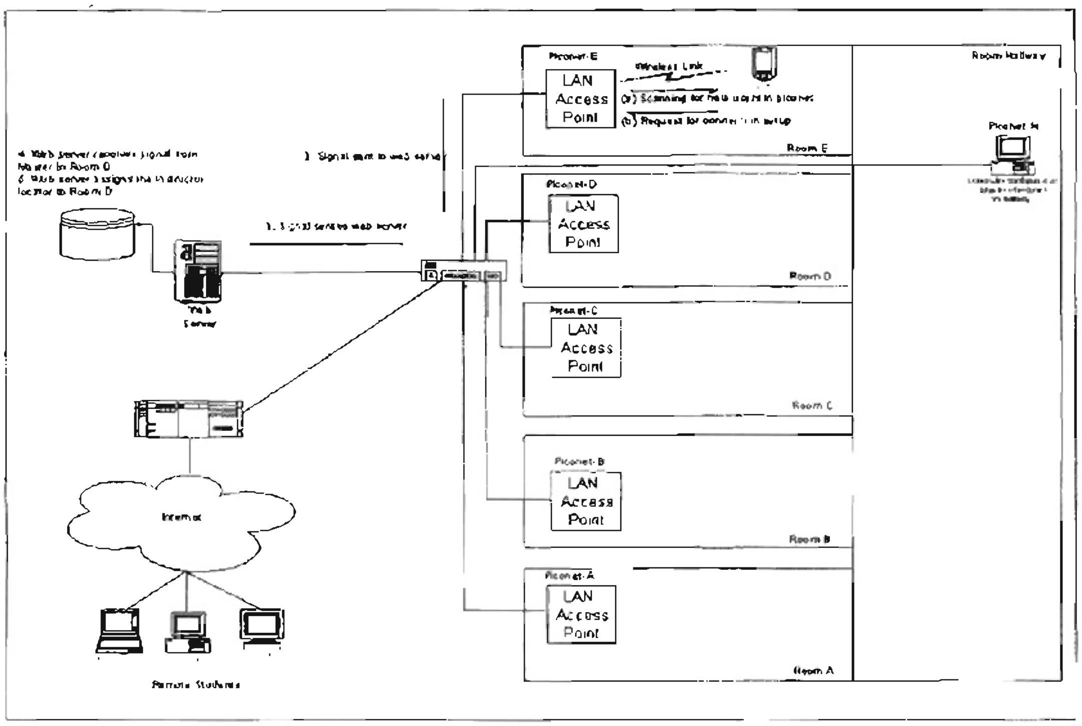

Piconet-E
Wireless Link
LAN Access Point
(a) Scanning for new users in piconet
(b) Request for connection setup
Room E
Room Hallway
Piconet-H
4. Web server receives signal from Master in Room D
5. Web server assigns the instructor location to Room D
3. Signal sent to web server
1. Signal sent to web server
Web Server
Piconet-D
LAN Access Point
Room D
Piconet-C
LAN Access Point
Room C
Piconet-B
LAN Access Point
Room B
Piconet-A
LAN Access Point
Room A
Internet
Remote Students

Fig 3.1. Implementation of Virtual Lab.

Instant Text Messaging:

This feature allows the student to send text messages to the Instructor from his/her desktop. The messages are displayed on the PocketPC carried by the Instructor.

Wireless AV Headset System:

The transfer of real-time audio and video (AV) from the instructor to students will be made possible by the development of the wireless AV headset for the instructor and is implemented as shown in Fig. 3.2.

This WI AV system enables the instructor to show procedures and facilities that may not be accessible to the students due to many reasons, which may include:

- the facilities only allow a very small number of people to approach at a time, and having all students come and see in turn may not be possible due to time limitations,

• the facility may be limited to trained or authorized personnel only,

• necessary changes/repairs to a system module were made which are currently a part of the ongoing experiments that need to be urgently informed to the students,

• and the obvious case where the DL students are remotely located from the actual lab and its facilities.

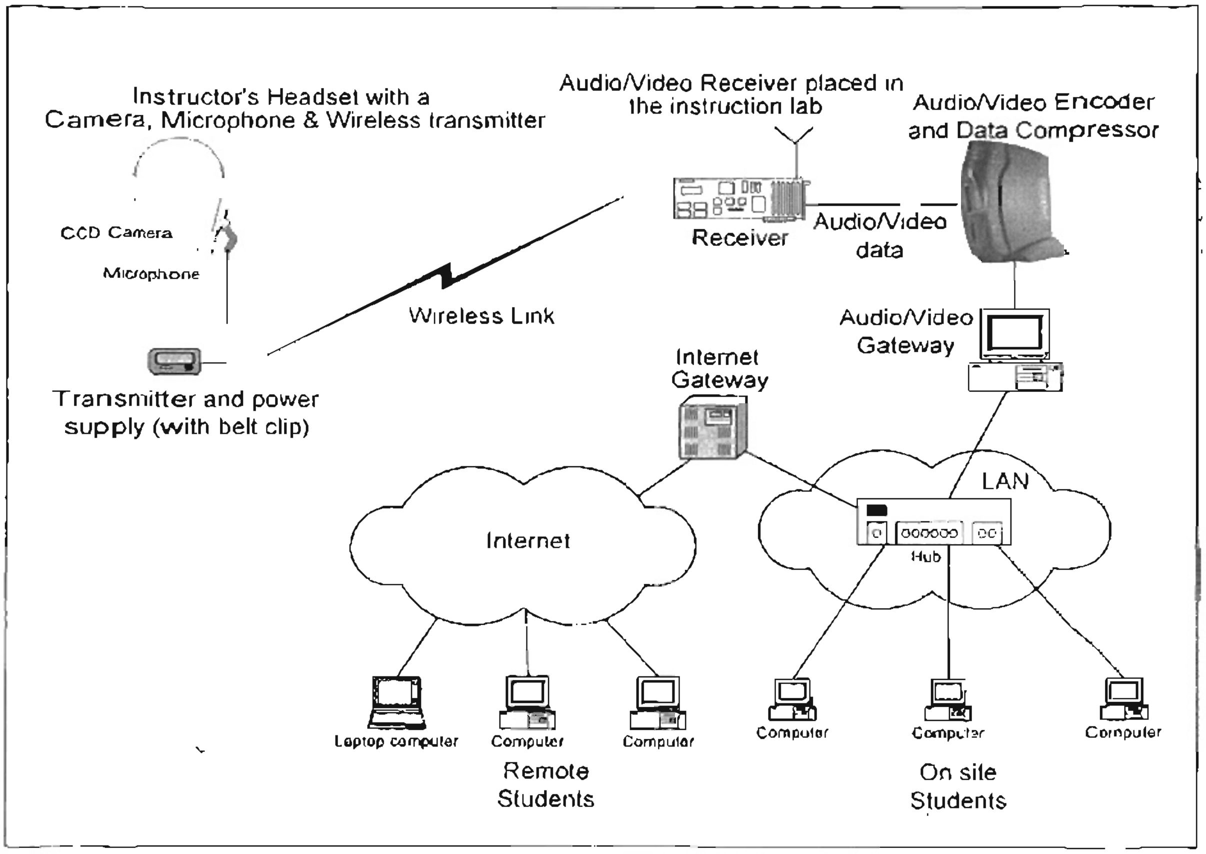

Instructor's Headset with a Camera, Microphone & Wireless transmitter
CCD Camera
Microphone
Transmitter and power supply (with belt clip)
Wireless Link
Audio/Video Receiver placed in the instruction lab
Receiver
Audio/Video data
Audio/Video Encoder and Data Compressor
Audio/Video Gateway
Internet Gateway
Internet
LAN
Hub
Laptop computer
Computer
Computer
Computer
Computer
Computer
Remote Students
On site Students

Fig. 3.2. Wireless AV Headset Solution.

3.2 Folder Structure used by the Wireless Instructor System on the Server

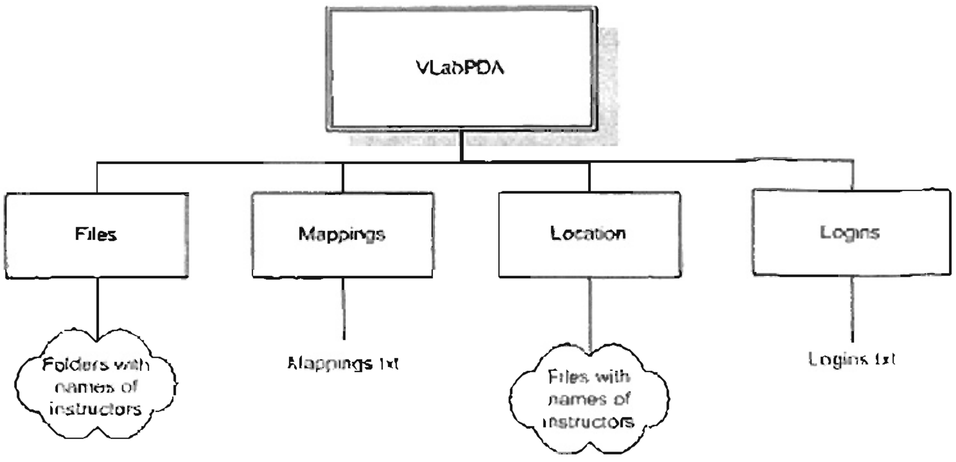

VLabPDA
Files
Mappings
Location
Logins
Folders with names of instructors
Mappings.txt
Files with names of instructors
Logins.txt

Fig 3.3. Folder structure on Server.

"VLabPDA" is the main folder on the server. It holds all the subfolders that are used by the Wireless Instructor System. This is the default name given to the Main folder but it could be changed to any other name if the changes are also added in Configuration utility.

The Main folder contains the folders "Files", "Mappings", "Location" and "Logins", as shown in Fig. 3.3, and whose names should not be changed. The "Files" folder contains subfolders which are named with the Instructors logged in at that instant. Each folder contains the files pushed by that particular instructor for the students. The "Mappings" folder contains a file called "Mappings.txt", which contains information mapping the Bluetooth address of each LAN access point with the room in which it is placed. The "Location" folder consists of files named with the Instructors logged in at that instant. These files contain the Bluetooth address of the LAN access point to which the instructor is connected along with the room where the LAN access point is located and the IP address assigned to the PocketPC. The "Logins" folder contains a file named "Logins.txt", which contains names of instructors who have access to the Wireless Instructor system along with their user names and passwords. This file is encrypted for security purposes.

Files of significance on the server:

Instructor Location files:

These files are located in the "Locations" subfolder and have names of the Instructors who are currently logged into the Wireless Instructor System. These files have the Bluetooth address of the LAN Access Point (LAP) to which the Instructor's PocketPC is connected to and the room in which it is along with the IP Address assigned to the PocketPC. The name of the Instructor is picked up from the "Logins.txt" file and the room name is picked up from the "Mappings.txt" file.

Example:

Filename: Dr.Jong-Moon Chung

Contents:

```
0x00408c588ba4      A       139.78.79.165
```

The server picks up information about the Instructor Location from these files and displays their location as dots on the map of the lab.

Mappings.txt:

This file holds information mapping the Bluetooth addresses of the LAN Access Points with the room in which they are present. The Addresses are listed one after other following the format

<Bluetooth address in hexadecimal format> tab space <Room name>

Example:

0x00408c588ba4	A
0x00408c588b95	B

This information is used by the Wireless Instructor Program to get the room in which the LAN Access Point to which it has made a connection is present. This information is used in the Instructor location files.

Logins.txt:

This file has information of all the Instructors who have access to the Wireless Instructor System. The information includes the Instructor's full name, user name and user password used to login to the Wireless Instructor program. These data are stored in the file in the following format

<Instructor name> tab space <Username> tab space <Userpassword>

But all the information is encrypted by XOR-ing them with a particular code byte.

3.3 Folder Structure used by the Wireless Instructor System on the PocketPC

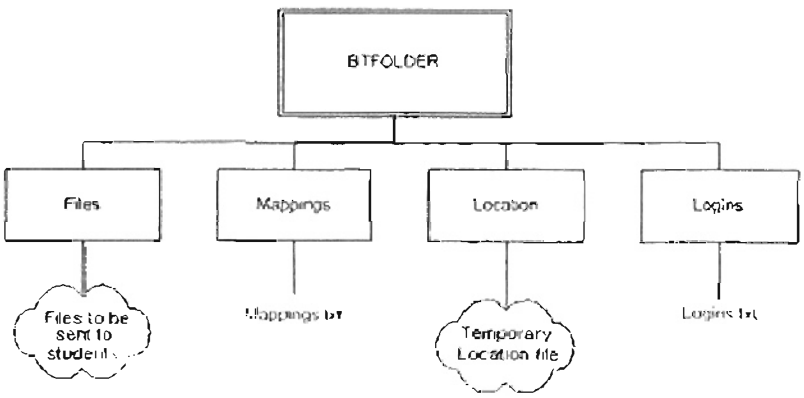


Fig 3.4. Folder structure on PocketPC.

This figure explains the folder structure used by the wireless instructor system on the PocketPC. "BTFOLDER" is the main folder on the server. It contains the folders "Files", "Mappings", "Location" and "Logins" , as shown in Fig. 3.4 and the names of these folders as well as the Main folder should not be changed. The "Files" folder may be used by the Instructor to store files that are intended to be transferred to the students. The "Mappings" folder contains a file called "Mappings.txt", which contains information mapping the Bluetooth address of each LAN access point with the room in which it is placed. This file is downloaded from the server using the Configuration utility. The "Location" folder is used to create the Location file locally named with the Instructor who is logged into the Wireless Instructor program before sending it to the server. This file contains the Bluetooth address of the LAN access point to which the instructor is connected along with the room where the LAN access point is located and the IP address of the PocketPC. These files are first created on the PDA and are then transferred to the server by FTP. The "Logins" folder contains the file named "Logins.txt",which is downloaded from the server and contains names of instructors who have access to the Wireless Instructor system along with their user names and passwords. This file is encrypted for security purposes.

3.4 Software Components of the Wireless Instructor System:

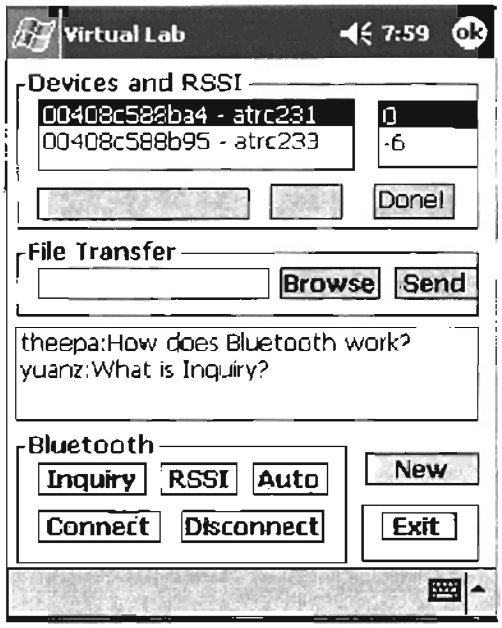
The Wireless Instructor system has three software components

1. Wireless Instructor Program

2. Instructor Access Management utility

3. Configuration Utility

Wireless Instructor program:

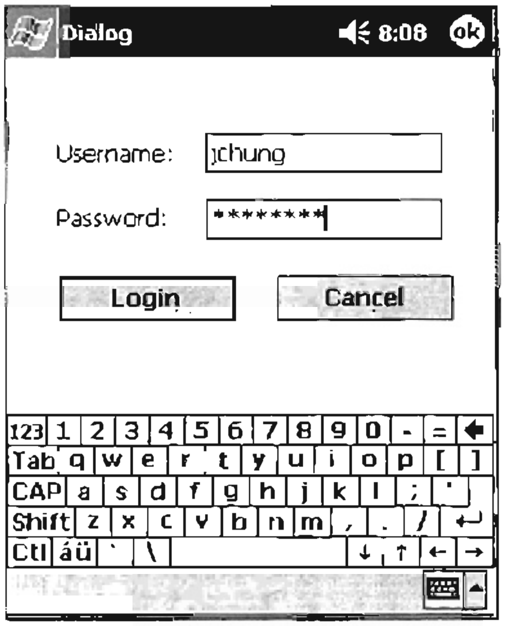


theepa:How does Bluetooth work?
yuanz:What is Inquiry?

Fig 3.5. The Wireless Instructor Program.

The Wireless Instructor program, the main window of which is depicted in Fig.3.5, is the main program which does Instructor Location, Web Pushing and displaying paging messages to the Instructor and runs on the PocketPC. This program plays the role of managing the Wireless connection of the PocketPC with the LAN Access Points and consequently with the LAN.

Not everybody has access to this program. "Logins.txt" holds the Instructor names, Login Names and passwords for the Instructors who have access. On starting the program a login page, as shown in Fig. 3.6 opens up where the Login name and password has to be entered. The program checks for the Login name and the corresponding password in the "Logins.txt" file and logs into the main window of the program only if the information entered is present and valid.



Username: jchung

Password: ********

Fig 3.6. Login Window of the Wireless Instructor Program.

The implementations of the functionalities of the program are explained in detail in the coming sections.

Instructor Access Management Utility:

This program is used to add, remove and edit Instructors' names, the usernames and passwords and runs on the desktop only. Only those who know a certain Login name and password - which is hard coded into the program - can access this program. This security feature has been added so that not everybody can add or delete access. Only the Instructors who are added by this utility will have access to the Wireless Instructor program. The details of the Instructors who have access are placed in a file named "Logins.txt" and the file is updated on the server in the location "\VLabPDA\Logins\". This file is encrypted by XOR-ing the information with a certain code. This file needs to be downloaded to the PocketPC before the Wireless Instructor Program to update the Login Information on the PocketPC. The file is downloaded by using the Configuration Utility. To know more about the utility please refer to Appendix B.

Configuration Utility:

This program is used to input all the basic information needed by the Wireless Instructor Program to perform its functions. All the information that is input is stored in a file named "VLab_Config.txt" in the local folder, "\My Documents\BTFOLDER\Configuration\". The data that are written to this file includes the IP Address of the server, the login name, login password, the Main parent folder on the server for the Wireless Instructor System which by default is "VLabPDA" and the name of the connection setting that should be used by the connection manager. The Wireless Instructor Program picks up all these information from the file. In addition to the above information that could be input, the program also downloads the files, "Mappings.txt" and "Logins.txt" from the server locations, "\VLabPDA\Mappings\" and "\VLabPDA\Logins\" respectively by using an FTP connection and places them in local folders, "\My Documents\BTFOLDER\Mappings\" and "\My Documents\BTFOLDER\Logins\" respectively. These files are also used by the Wireless Instructor System and hence the Configuration Utility needs to be run before starting the Wireless Instructor

Program if any changes have been done to Mappings.txt, Logins.txt or to the basic information of the server mentioned earlier. Please refer to Appendix A for more details on the Configuration Utility.

3.5 Wireless Instructor Program Implementations

Instructor location:

Instructor location is a feature in the Wireless Instructor system which gives information on which room of the Lab the Instructor is, at any instant. The Instructor carries a Compaq iPAQ PocketPC which is connected to the LAN wireless through Bluetooth. Each room has a LAN Access Point (LAP) to which the PocketPC makes the Bluetooth connection.

A text file on the Server contains information mapping the Bluetooth Addresses of all the LAP's used in the lab to the names of the corresponding rooms they are placed. This file named "Mappings.txt" is located in the folder,"/VlabPDA/Mappings/" on the server. This file is to be downloaded into the PocketPC into the folder "\My Documents\BTFOLDER\Mappings\" before the Wireless Instructor Program is started. A separate Configuration utility has been created which downloads the file automatically from the server to the PocketPC and places it in the folder specified above. Obviously this utility has to be run before starting the Wireless Instructor program so that updated information of the mappings is available.

The Bluetooth Software Development Kit (SDK) and the associated host stack to implement Bluetooth operations from the Wireless Instructor program were purchased from a company named ImpulseSoft. It provides lot of API's that could be used for basic Bluetooth operations but a TCP/IP connection may not be initiated with the API's from ImpulseSoft, Inc..

The Wireless Instructor program could me made to run in a "Manual Selection Mode" or "Automatic Selection Mode."

In Manual Selection Mode the Instructor could inquire for all devices in the locality by touching the button "Inquiry". The program lists the Bluetooth Addresses of 3 nearby LAP's along with the corresponding rooms they are located. The Instructor has an option to manually select each device and check for its Received Signal Strength Indicator (RSSI) value by touching on the button named "RSSI". The RSSI value gets displayed beside each device that was tested for RSSI. Then the instructor could select the device that he/she wishes to connect and touch on "Connect". The program writes the Bluetooth address and the Data Link Channel (DLC) of the device selected into the "HKEY_LOCAL_MACHINE\BuiltIn\BTSerialCE2\BDAddr" and Channel (DLC) in "HKEY_LOCAL_MACHINE\BuiltIn\BTSerialCE2\Dlc" keys of the registry respectively. This change in the registry values should not be made till the previous connection is broken completely. Then the connection manager API is called to create a Bluetooth connection with the selected LAP. Once the connection is established it creates a file in the name of the Instructor who has logged into the program, locally in the folder

"\My Documents\BTFOLDER\Location\". The file contains the Bluetooth Device address of the LAP to which the connection has been made, the room at which the LAP is present and the current IP address assigned to the PocketPC. This file is sent to the folder,"/VLabPDA/Location" on the server by creating an FTP connection to the server. The Manual Selection Mode could be used by the instructor if he/she is not going to move out of a particular room.

The flow of procedures for making a Bluetooth connection is given in Fig 3.7. The Wireless Instructor Application inquires for all the LAP's nearby. When a particular LAP is selected, the necessary changes are made in the registry as mentioned earlier. Windows CE provides a Connection manager which is capable of creating a TCP connection over Bluetooth by using Point to Point Protocol (PPP). The Application invokes the WinCE Connection Manager which dials out information for making the connection through the Virtual Serial Port 8. These data are captured by the Serial port driver and sent to the Bluetooth stack. The Bluetooth stack, on receiving the commands picks up the destination address, to which the connection is to be made from the registry, which was updated by the Wireless Instructor Application. The Bluetooth stack then invokes paging to make a connection with the destination LAP. On successful paging a Bluetooth connection is made after a Master/Slave switch. Then the PocketPC integrates with the LAN through the Bluetooth connection with the LAP.

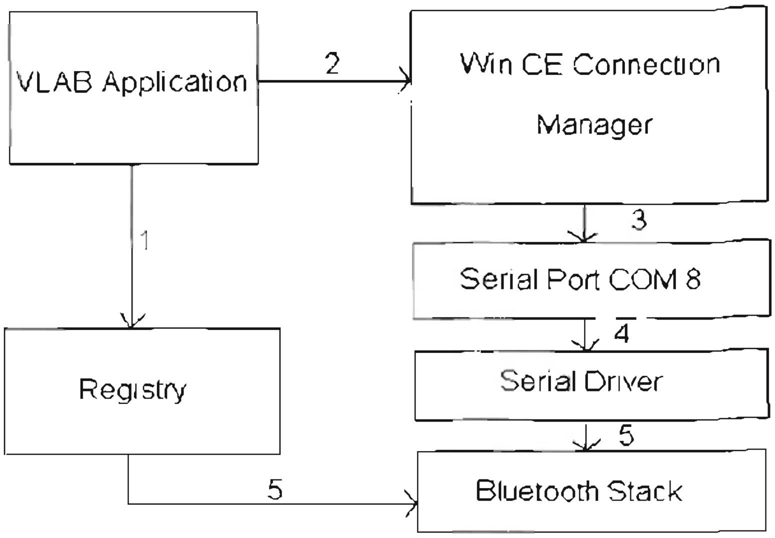


Fig 3.7. Procedure used by Wireless Instructor Program to pass BT Address and to make connection.

Procedures 3, 4 and 5 take place internally and are not controlled by the user developed application.

The Automatic Selection Mode can be started by touching on the "Auto" button. The Wireless Instructor program automatically inquires for all nearby LAP's once every minute. Along with inquiry it also measures the RSSI values of the devices. And it automatically connects with the LAP with the highest RSSI value by writing the Bluetooth address and the Data Link Channel (DLC) of the device into the "HKEY_LOCAL_MACHINE\BuiltIn\BTSerialCE2\BDAddr" and Channel (DLC) in "HKEY_LOCAL_MACHINE\BuiltIn\BTSerialCE2\Dlc" keys of the registry respectively and calling the Connection manager API to create a connection. On connecting to the LAN, the PocketPC creates a file containing the Bluetooth Device address of the LAP to which the connection has been made, the room at which the LAP is present and the current IP address assigned to the PocketPC in the local folder "\My

Documents\BTFOLDER\Location\". This file is sent to the folder,"/VLabPDA/Location" on the server by creating an FTP connection to the server. The whole process described above takes place once every minute. Thus the PocketPC refreshes its connection with the nearest LAP and updates the Instructor location every minute. The procedure is depicted as a flowchart in Fig. 3.8.

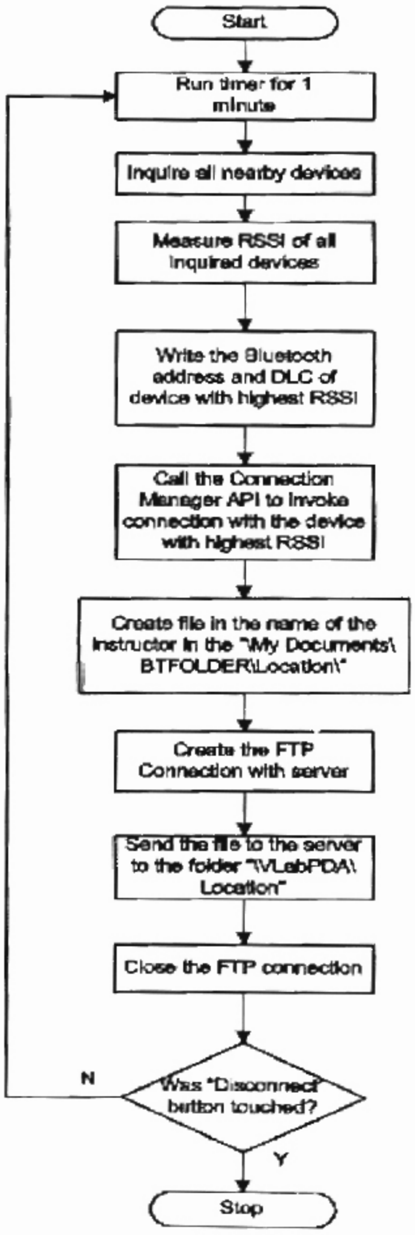

Start
Run timer for 1 minute
Inquire all nearby devices
Measure RSSI of all inquired devices
Write the Bluetooth address and DLC of device with highest RSSI
Call the Connection Manager API to invoke connection with the device with highest RSSI
Create file in the name of the instructor in the "My Documents\ BTFOLDER\Location\"
Create the FTP Connection with server
Send the file to the server to the folder "\VLabPDA\ Location"
Close the FTP connection
Was "Disconnect" button touched?
N
Y
Stop

Fig 3.8. Flowchart of the Automatic Selection mode.

Web Pushing:

The instructor could select a file and send it to a particular folder in the server using the program. On touching the "Browse" Button a File Browse Window is opened. The Instructor is able to select the file to be transferred to the server by touching it. The Browse Window closes once the file to be transferred is touched. The file name is displayed in an Edit box. The "Send" button is touched to send the file by creating a FTP connection. The ScottyFTP API from Ruksun Software based in Pune, India is used for programming the FTP operations.

The function ScottyFtpConnect is used to connect to the server which takes a FTP handle and IP address of the server as inputs.

After this, ScottyFtpLogin is used to login to the server by passing the login name and password as inputs to it.

ScottyFtpChangeDirectory is used to change the current directory to "/VLabPDA/Files/<Instructor Name>" on the server.

Then ScottyFtpPutFile is used to transfer the selected file, by reading its name from the Edit box mentioned earlier, to the "/VLabPDA/Files/<Instructor Name>" directory on the server.

The FTP connection is disconnected using ScottyFtpQuit.

The flow of procedures for Web pushing is shown in Fig. 3.9.

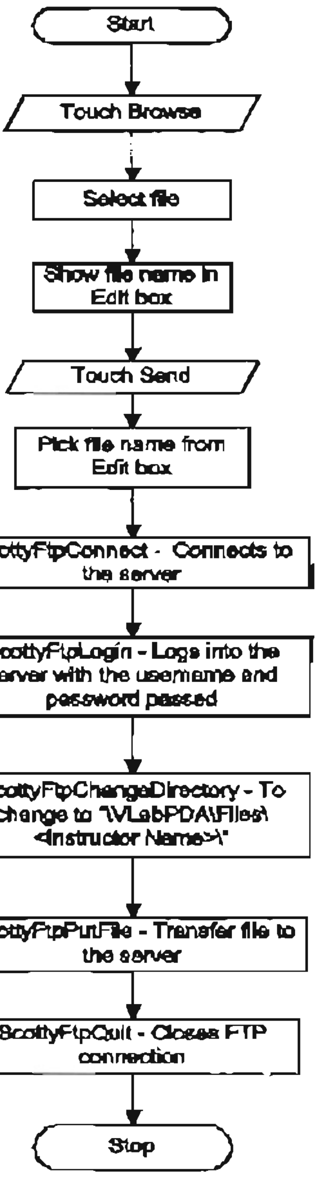

Start
Touch Browse
Select file
Show file name in Edit box
Touch Send
Pick file name from Edit box
ScottyFtpConnect - Connects to the server
ScottyFtpLogin - Logs into the server with the username and password passed
ScottyFtpChangeDirectory - To change to "\VLabPDA\Files\<Instructor Name>\"
ScottyFtpPutFile - Transfer file to the server
ScottyFtpQuit - Closes FTP connection
Stop

Fig 3.9. Flowchart of Webpushing.

The actual code implementation of the usage of the API from Ruksun could be looked up at Appendix C.

Paging of the Instructor:

The webpage presented to the students is added with a space for the student to enter text messages to be sent to the instructor. The student is able to select which Instructor the message is to be sent. The text message along with the student's login name is stored on the server. The Compaq iPAQ PocketPC connects to the server periodically every 30 seconds and retrieves all text messages sent to a particular instructor. The PocketPC makes a HTTP connection to the server and invokes a "php script" on the server which filters out the messages for the particular instructor and sends them to the PocketPC as shown in Fig. 3.10.

The messages are displayed on the Pocket PC in the
<Student Login name> : <Message>
format one after the other in an Edit box.

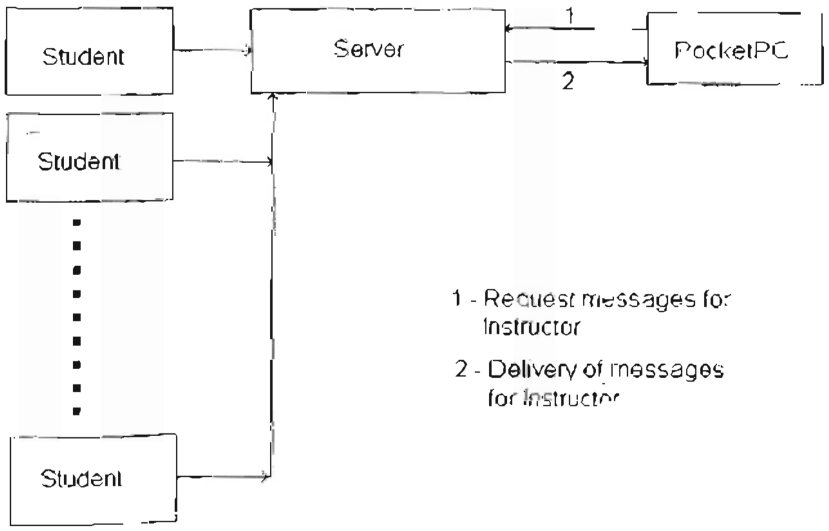

Student
Server
PocketPC
1
2
Student
Student
1 - Request messages for Instructor
2 - Delivery of messages for Instructor

Fig 3 10. Procedure for paging Instructor

3.6 Transfer of Audio and Video to students

An Audio/Video Headset is to be worn by the Instructor which transmits data wireless to the server which could then relay real-time video and audio on the webpage for the Virtual Lab on demand. The Audio/Video transfer is accomplished by using the following components from Microtek, Inc..

- Color CCD Camera with Pinhole Lens
- 500 mW, 900MHz, Audio/Video Transmitter and Receiver
- Mini Microphone with Built-in Amplifier
- 12V 8AA Battery Holder
- 2 Power "Y" Cables

as well as a Analog to USB converter named WinTV USB from Hauppage, Inc.

The Audio Video Headset transmits FM signals at 900MHz and hence it does not interfere with Bluetooth that is used by the PocketPC to communicate with the LAN.

Chapter IV

Description and analysis of the new proposals for Bluetooth Handover

Support for Bluetooth devices is being provided in Airports and in big offices. When the user moves around the place, uninterrupted traffic flow is to be provided which can be accomplished only if the device is able to lose the weak links due to the user's mobility and create new stronger links with nearby Access Points. This chapter proposes three new algorithms for Bluetooth Handover which have their own advantages and disadvantages and each suitable for a certain situation or application.

4.1 Initial Setup

4.1.1 Arrangement of Base Stations

The important consideration as with any mobile wireless communication system is to provide uninterrupted high quality connectivity. Among many factors that help towards this, the arrangement of the Base Stations is an essential part to be considered. The Base Stations need to be arranged in such a way that no area is left uncovered by the signal. The most famous arrangement of placing the Base Stations is in the middle of imaginary hexagonal cells, into which a large area can be divided into multiple cells based on the systems signal coverage range. This is the assumed model that will be applied in this paper, which is shown in Fig. 4.1.

The Base Stations denoted by the '+' sign are arranged in such a way to form imaginary hexagonal cells that cover the area required. The Additional Base Stations denoted by 'x' are added such that an extra number of devices could be served since each

Base Station can serve only a maximum of 7 slaves at anytime.

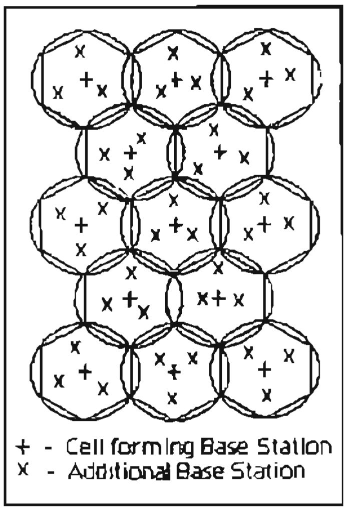


Fig. 4.1. The ideal arrangement of Base Stations for full coverage of area.

4.1.2 Features of the Base Station

For the handover operations to be possible using the Bluetooth technology, certain features need to be accommodated into the design.

1) The Base Stations have to be connected to a LAN/WAN network with gateways functioning as an interface between the networks.

2) The Base Station needs to have the signal power measuring feature to measure the power level of the link between itself and a particular Slave which is denoted by the RSSI variable, which is a feature in the Bluetooth specifications [5].

3) All Base Stations have to periodically enter Inquiry scan amidst the traffic between other slaves so as to be discoverable to new Bluetooth devices.

4) When a new link is brought up and traffic to a particular device is to be routed through that link, from then on, the information should be updated in the gateway such that all other devices and the external network can access the desired device.

The above features are required such that the Base Stations can communicate between themselves and assist the handover operations that are developed in this paper.

4.1.3 Initial Connection Setup

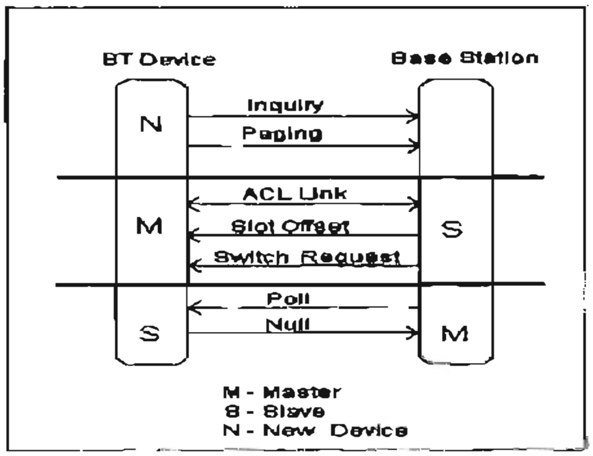


Fig. 4.2 Initial Connection Setup.

A connection between the Base Station and the Bluetooth device could be initiated by either system. In the case where the Base Station inquires periodically to find new Bluetooth devices in the piconet range, the Bluetooth devices need to wait until the Base Station allots time for itself to enter the Inquiry stage while it has already engaged in traffic transfer between slaves connected to it. This means that the user needs to wait for quite a while before getting connected to the Base Station. This is undesirable, thus an alternative second option of the Bluetooth device initiating the connection to the Base Station is provided. In this option, the Base Station would become a slave to many Bluetooth devices, which means it has to synchronize with each of them for each slot it communicates with a different slave. This requires additional system operational features, which most likely will result in a higher device cost. Thus, this option is not desired. Therefore, a combination of both techniques is proposed and described below. The procedure is also depicted in Fig. 4.2.

1) The Bluetooth device is in an inquiry procedure when it is not connected to any

Master

2) When it enters a piconet range, the inquiry from the Bluetooth device would evoke a response from the Base Station.

3) The Base Station responds with a frequency hopping sequence (FHS) packet, which includes its Bluetooth address.

4) On receiving the FHS packet, the Bluetooth device then enters the Paging mode and pages the Base Station with the address.

5) Once the Base Station responds to the page, a connection is setup between the device and the Base Station.

6) Then, the Base Station transmits link management protocol (LMP) messages to initiate the Master/Slave Switch.

7) The Slave and Master then switch roles and the Base Station becomes the Master and the wireless Bluetooth device the Slave.

4.2 Handover Proposals

4.2.1 Proposal 1

The first proposal is a procedure where the Bluetooth device periodically inquires while staying connected and maintains a stack of nearby Base Stations and chooses the Base Station to connect to when Handover is initiated as shown in Fig. 4.3.

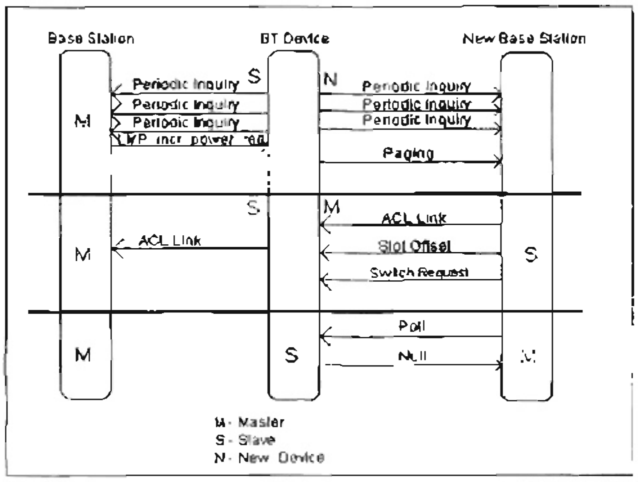


Fig. 4.3. Handover according to Proposal 1.

1) The Bluetooth device is always in "Periodic Inquiry" mode with a higher Inquiry period (say, 60.16 s) and looks for new Base Stations around when it is connected to a Base Station as a Slave for an Inquiry length of say, 1.28 s.

2) The Bluetooth device maintains a stack of the Base Station device addresses, it had found in its vicinity.

3) The Base Station device addresses found in the latest inquiry are added and arranged on top of the stack according to the RSSI values, the one with the highest RSSI value being the topmost.

4) When the RSSI of the link between the Bluetooth device and the current Base Station to which it is connected falls below the lower threshold level, the Base Station immediately sends a "LMP_incr_power_req" message to the Slave.

5) On reception of the LMP message, the Slave either increases its power level or in the case where it has already reached its maximum power level or if the feature is not supported, it starts paging the Base Stations one by one using the stack of Base Station addresses starting with the topmost.

6) The device then connects to the first Base Station (as a Slave), which responds

to the Paging. If a connection was not able to be made with the first device it will choose the next one and so forth.

7) If none of the Base Stations in the stack of addresses respond, then the device has to do an Inquiry and find a nearby Base Station and connect to it. If a connection is made, the mobile unit will be the Master and the Base Station will be the Slave since the mobile unit initiated the Inquiry.

8) Then, the Base Station initiates a Master/Slave Switch after which the Bluetooth device becomes a Slave to the new Base Station.

9) The Bluetooth device stays connected with both the Base Stations but uses the new link for all traffic. It may start using the old link in case its RSSI value comes over the threshold level and the new link's RSSI value goes lower than the threshold.

10) It disconnects any one of the links when its supervision timer times out. Mostly, it is the unused link that is disconnected by the link supervision timeout as the active link is constantly monitored by using RSSI.

4.2.2 Proposal 2

In the second proposal, the Base Station keeps track of the RSSI of the Slave unit, and when the RSSI goes below an acceptable level a request is initiated to the nearby Base Stations to connect to the wireless device as shown in Fig. 4.4.

1) The Base Station measures the RSSI level at regular intervals.

2) When the RSSI value falls below the lower threshold level, the Base Station sends "LMP_incr_power_req" message to the Bluetooth device.

3) This LMP message requests the device to increase its power level. If the device is capable of that, it increases its power.

4) In case the device has reached its maximum power level or does not have the capability to increase its power, it enters into the "Page scan" mode while maintaining the traffic flow connection to the Base Station.

5) The Base Station also requests all nearby Base Stations, through the wired network, to page the device. It also sends the device's Bluetooth address and its slot offset to speed up the paging process. It may achieve this by a broadcast of the packet to all the Base Stations in the local physical network.

6) On detection by a new Base Station, the device gets connected to it as a Slave and uses the new link for all traffic until it is of good signal strength than the old link and above the threshold level.

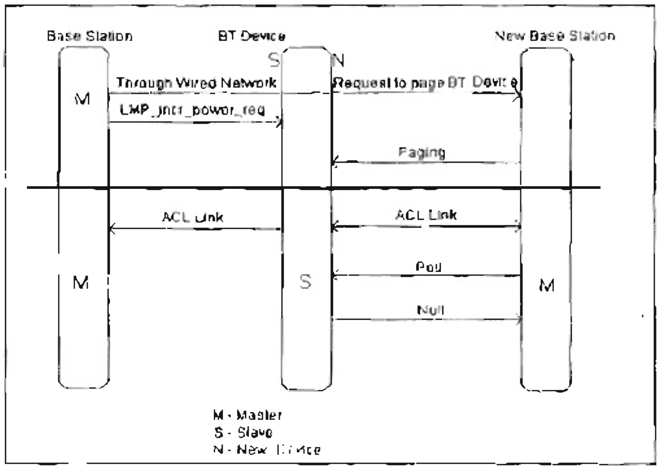


Fig. 4.4. Handover according to Proposal 2.

7) The device maintains both the links until either one of them disconnects by a link supervision timeout. Same as is in Proposal 1, commonly it is the unused link that is disconnected by the link supervision timeout as the active link is constantly monitored using the RSSI.

4.2.3 Proposal 3

The third proposal follows a very simple methodology of ensuring reliable connectivity of the Bluetooth device through a backup link maintained with another Base Station as shown in Fig. 4.5.

1) In the initial connection setup, while a Bluetooth device is newly introduced into the Base Stations' range, if the device finds two Base Stations in the range it connects with both immediately. If it finds more than two Base Stations it chooses the two Base Stations that responded first.

2) At any time, one of the links acts as a medium of transfer of traffic while the other is inactive. The inactive device could be just a active link that is not used for traffic or it could also be placed in the park mode.

3) The Base Stations monitor the RSSI level of all their connected links. When the RSSI of any link falls below the lower threshold value, it informs the Slave using an "LMP_incr_power_req" message.

4) If the Slave is capable of increasing its transmission power, it does that. If it is working in its full power or is not capable of increasing its power, it starts using the other good link for traffic. If the backup link was in park mode, then it is brought up to active link and this link is used for the traffic. The poorer quality link is disconnected. The backup link is also constantly monitored and if the link quality gets poor it is replaced with a better one.

5) The device goes into the "Periodic Inquiry" mode with Inquiry length and Inquiry period of say, 1.28 s and around 5.12 s, respectively. An alternative way may be to perform inquiry when the amount traffic is low, assuming prior

knowledge is obtained about it.

6) The Bluetooth Device inquires and connects as a Master with another new Base Station, in addition to the currently connected Base Station.

7) The newly selected Base Station then initiates a Master/Slave switch to make the wireless Bluetooth device its Slave. The new link is kept inactive or is not used for any traffic.

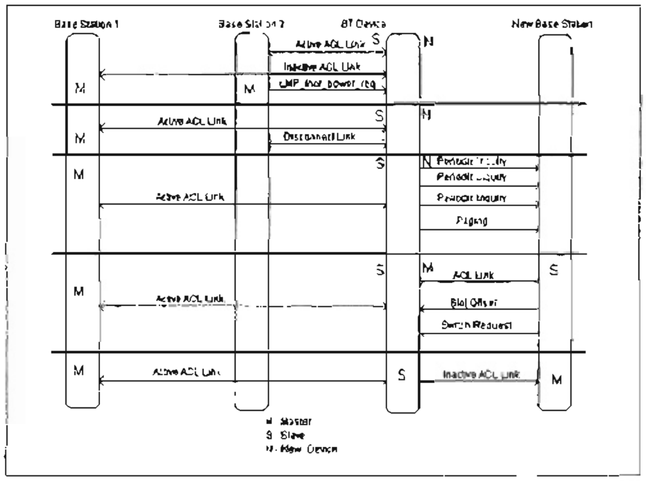


Fig. 4.5. Handover according to Proposal 3.

8) The Bluetooth device maintains both the links until the signal quality of one of the links' degrades below the required threshold. Once one of the links is dropped, the device again performs procedures to setup an alternative corresponding link.

4.3 Analysis of Handover

The proposed handover techniques have been implemented and analyzed for their timing performance. The timings for Inquiry and connection have been measured by developing an application with capabilities to time the desired events. Further software has been developed which implements the handover topologies proposed and analyses their performance.

4.3.1 Test Setup

Ericsson Bluetooth Development Kits™ were used as the Bluetooth Devices and Base Stations in building the Bluetooth wired and wireless network. The applications were implemented over the Ericsson Bluetooth Protocol Stack™. The CATC Merlin Bluetooth Protocol Analyzer™ was used to monitor the packet transfer between the devices and the Agilent Technologies E1852B Bluetooth Test Set™ was used to measure the signal quality.

4.3.2 The Timing Analysis

Handover Technique	Handover Timing (Range) (ms)	Handover Timing (Average) (ms)
Proposal 1	220 – 241	231.2
Proposal 2	170 – 190	172
Proposal 3	1.875 – 2.500	1.887

Table. 4 1. Minimum Handover Timings.

From the performed tests, the timing for Inquiry and connecting with a Service Discovery was measured and recorded. The tests were repeated 10 times and the time was recorded for each event. The results are represented in Table 4.1.

1) Range of time taken to discover a device: 481 – 4196 ms

2) Average time taken to discover a device: 2339.6 ms

In Proposal 1, after the connection setup, a Master/Slave switch needs to be conducted which takes 70 ms. The given timing is the minimum timing needed for Handover and this could increase if a connection was not able to be made with the first Base station in the stack and more base stations are tried for connection. The handover timings may also increase if the new Base Station has one or more SCO links which have higher priority than page scan and can interrupt it. Proposal 2 does not require Master/Slave Switch and is faster than proposal 1. The given timing is the minimum handover timing and it would increase if the new Base Station had one or more slaves with SCO links as they interrupt the paging. Moreover there may be delays with the Page request message to be transmitted from the old Base Station to the new Base Station. In the third proposal, a link already present is made active. So the transition is almost instantaneous though a backup link will leisurely be established again commonly occupying backup channel resources in support of the rapid handover. The timings given are taken by assuming that as soon as the LMP message is received by the slave, it changes its clock to the new Base Station and immediately receives a POLL packet from the new Base Station. Then it would respond in the immediate slot and inform that it would be its new Master. But in reality, if there are many slaves with the new Base Station the POLL packet to the slave would come later and hence the handover will be delayed.

4.3.3 The Ping-Pong effect Analysis

When a device lies almost equidistant between 2 Base stations and the RSSI balance between the two is fluctuating very rapidly – the Base station RSSIs become greater than each other alternately very rapidly – the handover procedure may be started at each reversal of the RSSI values. This is called as the Ping-Pong effect.

The Bluetooth handover algorithms were implemented with code to counter the Ping-Pong effect. The Handover is not started until 10 continuous samples of the RSSI value are below the threshold value that would start handover. The samples are taken with a gap of 1 ms between them. Apart from this, the measuring of RSSI values after a handover starts only after a timeout value, say 1 minute. After a handover takes place, the timer is started and till then the measurement of the RSSI value is stopped and restarted at the end of the timer. When 10 continuous samples of the RSSI value dip below the threshold level, handover will be started. Thus the Ping-Pong effect is avoided.

The general method of Hysteresis that is used to avoid the Ping-Pong effect is also applied as shown in Fig.4.6. The upper and lower threshold values for the Hysteresis need to be found as per the requirements of the application. If a device is connected to, say Base station 1 and is moving towards Base station 2 the Handover would take place from Base station 1 to Base station 2 when the signal level goes below the upper threshold. If in case, the signal power of Base Station 1 becomes greater than Base station 2 in the course of time but still within the upper and lower threshold levels, the handover will not take place. The handover from Base station 2 to Base station 1 will start only when the power level of Base station 2 drops below the lower threshold. Applying this technique avoids the Ping-Pong effect. But appropriate threshold levels have to be chosen for errorless functioning.

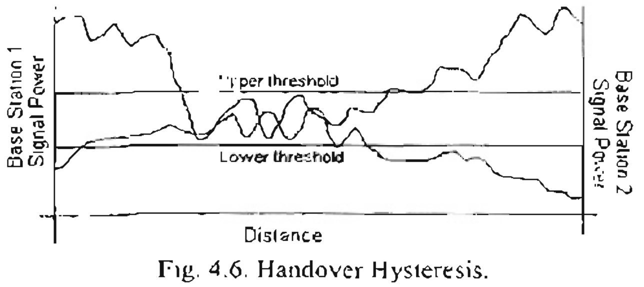


Fig. 4.6. Handover Hysteresis.

4.3.4 Analysis of interruption to traffic:

In the proposal 1, Inquiry takes place periodically during normal operation of the devices. The Inquiry is done by the slave so there may be circumstances, where when the master sends a packet, the slave may not receive it and respond to it. This could be avoided by starting the Inquiry when there is no traffic from the slave to be transferred to the master. When there is no data in the slave to be sent, a slave sends a NULL packet in response to the POLL packet from the master. Periodically the slave checks itself if it has data to send. If it has it waits till the master sends a POLL packet. In response to this, the slave sends the data and then waits for the next time the master sends a POLL. If there is no data it sends a NULL and immediately after that starts the Inquiry.

In proposal 2 and 3, there is no time lost in inquiry during traffic. In proposal 2, the data to be sent to the slave from the old Base Station may be buffered and sent to the new base station. Actually this data has to be sent to all the nearby Base Stations and the one that has connection to the slave sends it to it. In proposal 3, both Base Stations have knowledge of the other and hence when one the active link is broken, all the data from that Base Station are sent to the new active Base Station through the wired link. And hence there is no loss in traffic.

Chapter V

Conclusion

The Virtual lab was implemented successfully and the system has been performing very well. The instructor has found new freedom in moving around the rooms of the lab and instructing. In every lab course, when a particular exercise needs to be displayed, the students come in groups taking turns to view as to how it is done. The Audio/Video headset reduces the time lost in such activities. The students are able to see what the Instructor is working from their seats on their desktops. They could pose questions to the instructor by typing text messages that are displayed on the Instructor's PocketPC along with the student's login id to the Virtual Lab. The instructor saves a lot of time by being able to type in files and send them immediately to the students using the PocketPC. The primitive handover technology works without fault and does not interrupt during the presence of traffic. The Virtual Lab is a big step in infusing high-end technology for aiding education in Oklahoma State University. The Wireless Instructor system is shown in Fig. 5.1.

The first two proposals for Bluetooth Handover are techniques where the discovery of the devices is avoided during the failure stage and conducted before the need for handover arises. Although in the first proposal the traffic may be interrupted periodically for a small time for inquiry. The second proposed method is faster than the first as the Master/Slave switch is not required. The third proposed method is a technique where a backup link is available to replace the faulty link. Though this might be the fastest way to conduct handover, this topology will lead to a larger number of Base Stations in support of the same user population since each device at any time needs to maintain 2 links. The proposed techniques above can be used as a guideline to handover development in Bluetooth systems. The individual parts of each technique may be independently selected out and combined in establishing an improved handover technique that may suit a particular application or Bluetooth network configuration. The technique for handover may also be changed dynamically while roaming depending on the number of Bluetooth devices or Base Stations that need to be interactively discovered and the amount of traffic and the traffic service requirements.

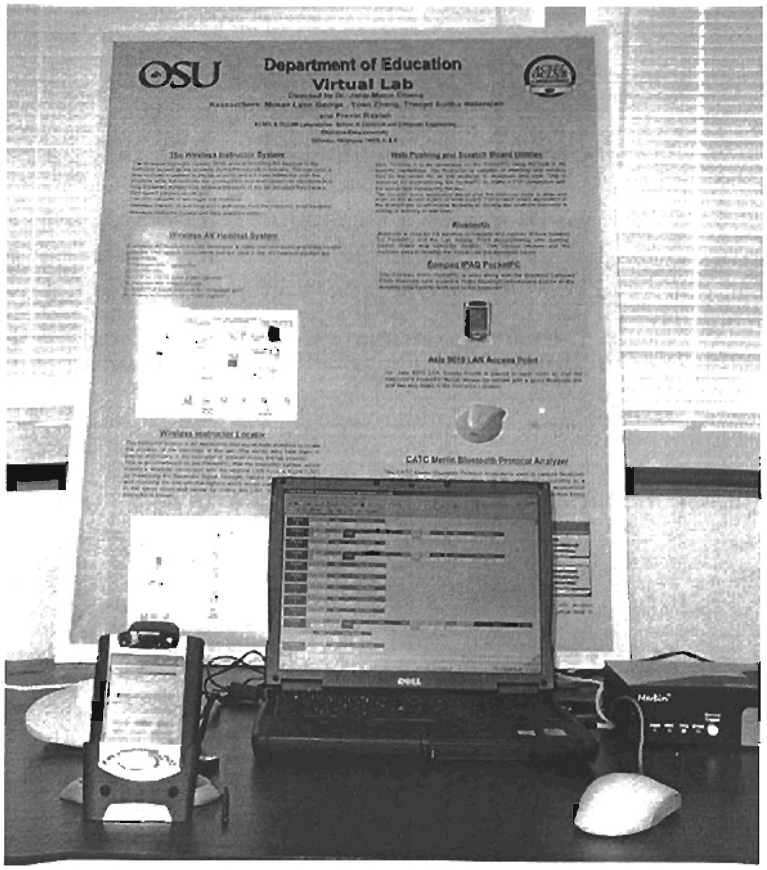


Fig 5.1. The Wireless Instructor System

REFERENCES

[1] C. C. Ko, B. M. Chen, S. H. Chen, V. Ramakrishnan, R. Chen, S.Y. Hu, and Y. Zhuang,"A large scale web-based virtual oscilloscope laboratory experiment," *IEE Engineering Science and Education Journal,* Vol. 9, No. 2, pp. 69-76, April 2000.

[2] C. C. Ko, B. M. Chen, S. Hu, V. Ramakrishnan, C. D. Cheng, Y. Zhuang, and J. Chen,"A web-based virtual laboratory on a frequency modulation experiment," *IEEE Trans. Systems, Man and Cybernetics,* part C, vol. 31, pp. 295 -303, Aug. 2001.

[3] M. Serra, E. Wang, and J. C. Muzio,"A multimedia virtual lab for digital logic design,"in 1999 *IEEE Int. MSE Conf.,* pp. 39-40.

[4] C. Rohrig, and A. Jochheim," The Virtual Lab for Controlling Real Experiments via Internet," *Proc. of IEEE Int. Symposium*, Aug. 22-27, 1999.

[5] S. Baatz, M. Frank, R. Gopffarth, D. Kassatkine, P. Martini, M. Schetelig, A. Vilavaara, "Handoff support for mobility with IP over Bluetooth," *Proc. IEEE LCN 2000*, pp. 143- 154, Tampa, Nov. 2000

[6] D. J. Y. Lee and W. C. Y. Lee , "Ricocheting Bluetooth," *Proc. IEEE 2nd Int. Conf. Microwave and Millimeter Wave Technology 2000*, pp. 432-435, 2000.

[7] D. Lee and W. Lee, "Integrating bluetooth with wireless and ricocheting," *Proc. 11th IEEE PIMRC 2000*, vol. 2, pp. 1310 -1314, 2000.

[8] M Arbrecht, M Frank, P Martini, M Schetelig, A Vilavaara, and A Wenzel, "IP Services over Bluetooth: Leading the Way to a New Mobility," *Proc. IEEE LCN'99*, pp. 2-11, Oct. 1999.

[9] J. Tourrilhes, *Bluetooth Roaming Proposals.* Basic Book/Monograph Online Sources, Oct. 9, 2000 (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Papers/apr-jt.pdf).

[10] Bluetooth SIG, *Specification of the Bluetooth System-Version 1.1* Specification volume 1, Feb. 2001.

[11] J. Bray and C. F. Sturman, *Bluetooth Connect without cables.* 1st ed., Upper Saddle River, N.J.: Prentice Hall PTR, 2001.

[12] P. Bhagwat and A.Segall, "A routing vector method for routing in Bluetooth scatternets," *Proc. Sixth IEEE Intl. Workshop on Mobile Multimedia Commun.* 1999, Nov. 1999.

[13] C. McDaid, *Routing connections in Bluetooth.* Basic Book/Monograph Online Sources, Apr.2001 (http://www.palowireless.com/bluearticles/cc3_handover.asp).

[14] A. Kansal, "Handoff in Bluetooth Public Access Networks," unpublished.

[15] M. L. George, L. J. Kallidukil, and J.-M. Chung, "Bluetooth handover control for roaming system applications," *Proc. of IEEE MWSCAS 2002*, Tulsa, Oklahoma, Aug. 4-7, 2002.

[16] Ruksun Software Technologies, *Programmer's Guide for Scotty FTP API for Windows CE.* Jan. 2001.

APPENDIX A

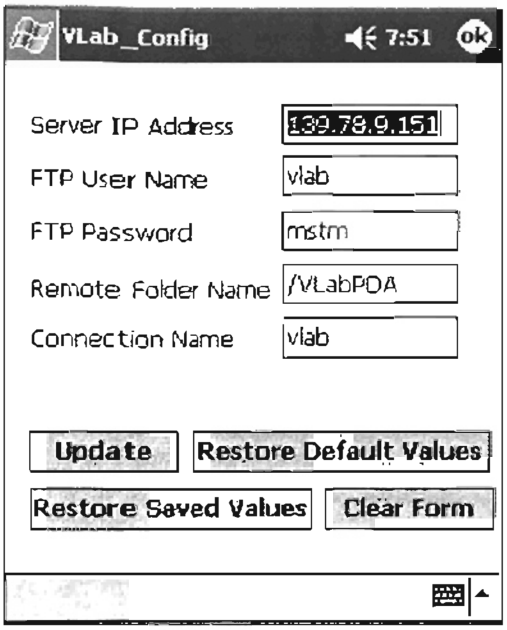
Description of the Configuration Utility

The Vlab configuration utility allows users to enter the Server IP Address, FTP user name, FTP Password, Remote folder name and Connection name and these values are stored on a file locally as well as download the Login and Mapping information. The utility also has an option to load default values in case the user wants to revert back to the default values. These values can be changed and updated in the file.

This utility is necessary because it is possible that the list of users who are given access to the Wireless Instructor system is prone to updating and so is the placement of the LAN access points in the various rooms of the lab. The Server's IP address may change when the network is altered. For security purposes, the FTP password may have to be changed. The Remote Folder Name and Connection Name may also be updated. One method that may be used is to store all this information on the server and update the PDA each time the application is used but is not possible to connect to the web server without knowing the FTP User Name and FTP Password. So this information about the connection and location of the files on the server are given to the PDA locally and the Login list and LAN Access Point-Room are updated manually when needed to.

The application updates the Pocket PC with the Login and Mapping files from the remote web server which are transferred and stored locally through an FTP connection.

The Login file contains the names, user names and passwords of the instructors who have access to the wireless instructor system. The Mapping file has information mapping Bluetooth addresses of LAN access points to the rooms in which they are placed. The Configuration Utility looks as shown in Fig. A.1.



Server IP Address	139.78.9.151
FTP User Name	vlab
FTP Password	mstm
Remote Folder Name	/VLabPDA
Connection Name	vlab

Fig. A.1. Configuration Utility window.

Server IP Address: This is the IP address of the web server in dotted decimal notation.

FTP User Name: This is the user name used to access the web server using File Transfer Protocol.

FTP Password: This is the password used to access the web server using File Transfer Protocol.

Remote Folder Name: This is the name of the Main root folder on the web server where all files related to the Wireless Instructor system are stored.

Connection Name: This is the name of the connection setting, configured to connect using Bluetooth Null Modem, used by the WinCE Connection Manager to connect to the LAN Access Points.

'Update' button: This button saves all the information entered in the utility in a local configuration file ("\My Documents\BTFOLDER\Configuration\VLab_Conf.txt") and also downloads the Login and Mapping information from the web server.

'Restore Default Values' button: This button loads the default values onto the Edit boxes in case the user wants to revert back to the default values.

'Restore Saved Values' button: This button loads the values that have been stored most recently in the configuration file onto the Edit boxes. When the utility is opened, the saved values are loaded automatically.

'Clear Form' button: This button is used to clear all the entries in the Edit boxes.

APPENDIX B

Description of the Instructor Access Management utility

Introduction

This program is a part of the software of virtual lab project for managing user accounts. It only permits administrator account login and does some operations for user accounts, such as adding users, deleting users, changing password, etc.

It gets the users' information from the FTP server, finishes the operation of administrator and then updates the users' information on the FTP server. It can be used on any desktop and is very easy to use.

How to use

1. Run the program, the login window will be showed first as in Fig. B.1.

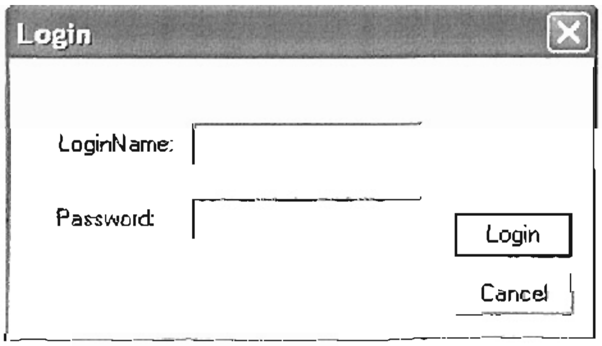
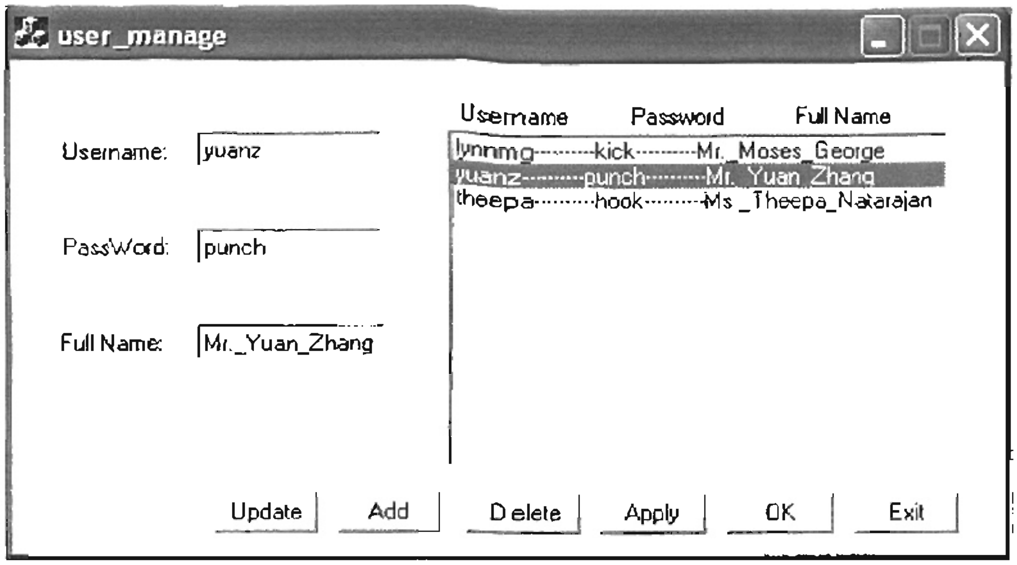


Fig B.1. Instructor Access Management Utility Login window.

Type the login name and password. Click *login* to access the program, clicking *cancel* will exit the program.

The main window is as in Fig. B.2.


user_manage

Username: yuanz

PassWord: punch

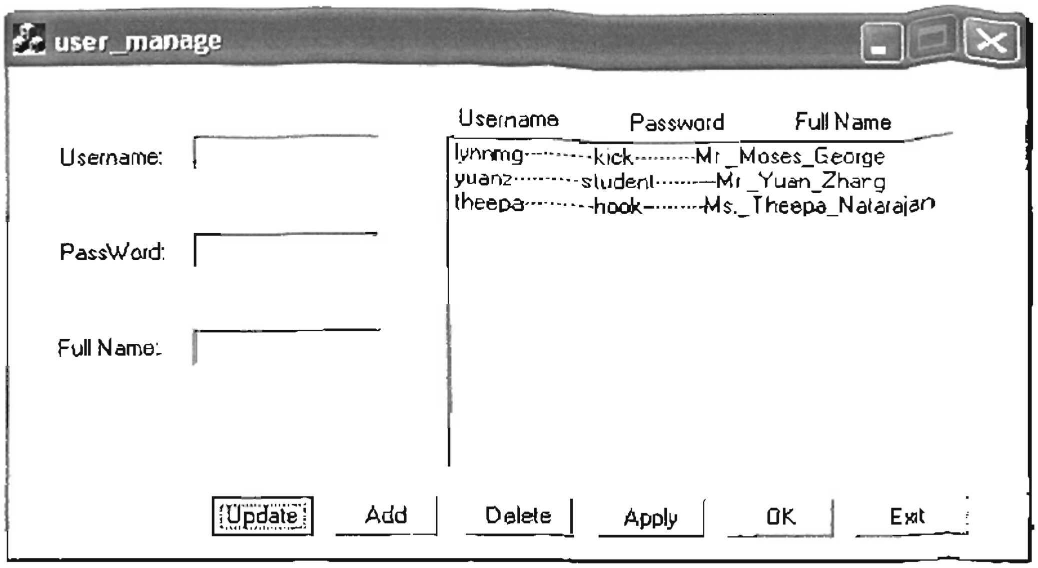
Full Name: Mr._Yuan_Zhang

Username	Password	Full Name
lynnmg	kick	Mr._Moses_George
yuanz	punch	Mr._Yuan_Zhang
theepa	hook	Ms_Theepa_Natarajan

Fig B.2. Instructor Access Management Utility window – Edit.

In the right window, the first column is the username, the second one is the password, and the third one is the full name of the user. Though the example shows names separated by an underscore, the names can be separated by a space.

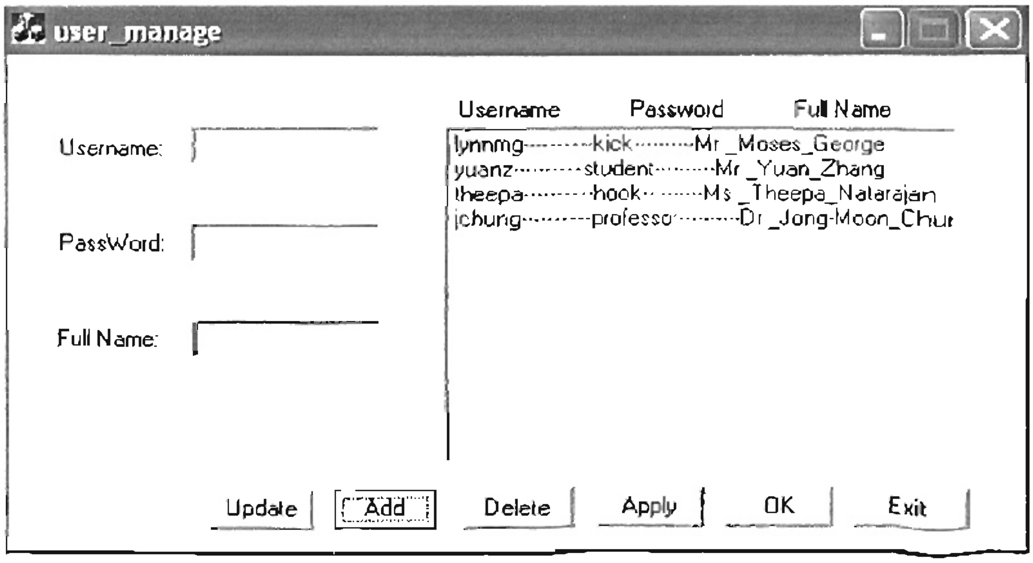
2. Choosing one user in the right window by clicking on it will show the information individually on the Edit boxes. Change the item that you want to change and then click *update*. The change made to that item is stored and is depicted in Fig. B.3.



```
Username        Password        Full Name
lynnmg---------kick---------Mr_Moses_George
yuanz----------student--------Mr_Yuan_Zharg
theepa---------hook---------Ms._Theepa_Natarajan
```

Fig B.3. Instructor Access Management Utility window – After Edit.

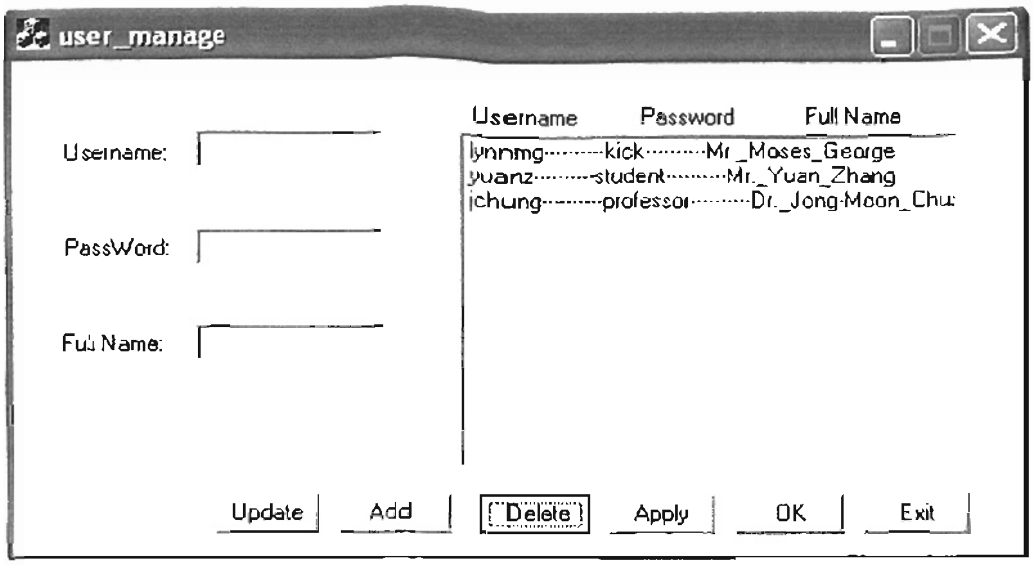
3. Write the username, password and full name in the left windows, click *add* and you can add a new user account and it is depicted in Fig. B.4.



```
Username        Password        Full Name
lynnmg---------kick---------Mr_Moses_George
yuanz----------student--------Mr_Yuan_Zhang
theepa---------hook---------Ms_Theepa_Natarajan
jchung---------professo---------Dr_Jong-Moon_Chur
```

Fig B.4. Instructor Access Management Utility window – After Add.

4. Choose one user account from the right window, and then click *delete*, you can delete the user that you want as depicted in Fig. B.5.



Username	Password	Full Name
lynnmg	kick	Mr._Moses_George
yuanz	student	Mr._Yuan_Zhang
jchung	professor	Dr._Jong-Moon_Chu

Fig B.5. Instructor Access Management Utility window – After Delete.

5. When you click *apply*, the program will store all user accounts that are in the right window to the FTP server. The directory is "/VLabPDA/Logins/Logins.txt".

6. When you click *ok*, the program will store all data to the FTP server and close the program.

7. When you click *exit*, the program will be closed without saving any change.

APPENDIX C

Code written for FTP connectivity over Bluetooth

```
/***************************************************************************
Function Name        : FTPOpen
Description          : This function opens the FTP connection and logs into the server
Input                : IP Address of server, Username, Password
Returns              : Boolean – True or False
***************************************************************************/
bool CUiDlg::FTPOpen()
{
        bFileTransferProgress = TRUE;

        FTPHandle = ScottyFtpCreate();
        if(FTPHandle == NULL)
        {
                bFileTransferProgress - FALSE;
                return FALSE;
        }

        if ( ScottyFtpConnect(FTPHandle,Server_IP Address) <0)
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

        if ( ScottyFtpLogin(FTPHandle, FTP_UserName, FTP_Password) <0 )
        {
                bFileTransferProgress - FALSE;
                return FALSE;
        }

        bFileTransferProgress = FALSE;
        bFTPPresence = TRUE;

        return TRUE;
}

/***************************************************************************
Function Name        : FTPSendFile
Description          : This function sends a file to the server
Input                : Remote Destination folder, Local file name, Destination file
name
Returns              : Boolean – True or False
```

```
*******************************************************************/

bool CUiDlg::FTPSendFile()
{
	m_Edit_Status.SetWindowText(_T("Busy!"));
	bFileTransferProgress = TRUE;

	if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}

	if ( ScottyFtpPutFile(FTPHandle, LocalFile, RemoteFile) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}
	m_Edit_Status.SetWindowText(_T("Done!"));
	bFileTransferProgress = FALSE;

	return TRUE;
}

/*******************************************************************
Function Name		: FTPGetFile
Description		: This function gets a file from the server to a local folder
Input			: Remote Source folder, Local Destination file name, Remote
Source file name
Returns			: Boolean - True or False
*******************************************************************/

bool CUiDlg::FTPGetFile()
{
	m_Edit_Status.SetWindowText(_T("Busy!"));
	bFileTransferProgress = TRUE;

	if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}

	if ( ScottyFtpGetFile(FTPHandle, RemoteFile, LocalFile) <0 )
	{
		bFileTransferProgress = FALSE;
```

```
                return FALSE;
        }

        m_Edit_Status.SetWindowText(_T("Done!"));
        bFileTransferProgress = FALSE;

        return TRUE;
}

/*****************************************************************************
Function Name           : FTPDeleteFile
Description             : This function deletes a file on the server
Input                   : Remote folder, Remote file name
Returns                 : Boolean – True or False
*****************************************************************************/

bool CUiDlg::FTPDeleteFile()
{
        bFileTransferProgress = TRUE;
        if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

        if ( ScottyFtpDeleteFile(FTPHandle, RemoteFile) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }
        bFileTransferProgress = FALSE;
        return TRUE;
}

/*****************************************************************************
Function Name           : FTPClose
Description             : This function closes the FTP connection with the server
Input                   :
Returns                 : Boolean – True or False
*****************************************************************************/

bool CUiDlg::FTPClose()
{
        ScottyFtpQuit(FTPHandle);
        ScottyFtpDestroy(FTPHandle);
        bFileTransferProgress = FALSE;
        bFTPPresence = FALSE;
        return TRUE;
}
```

APPENDIX D

Code for the Wireless Instructor program

Program Files:

CLoginDlg.cpp

```
// CLoginDlg.cpp : implementation file
//
#include "stdafx.h"
#include "resource.h"
#include "CLoginDlg.h"
#include "uiDlg.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define DEFAULTLOGINNAME "admin"
#define DEFAULTPASSWORD "virtualbluetooth"

/////////////////////////////////////////////////////////////////////////////
// CLoginDlg dialog


CLoginDlg::CLoginDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CLoginDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CLoginDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT

}

BOOL CLoginDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// TODO: Add extra initialization here
```

```
        m_Edit_Password.SetPasswordChar('*');
//      GetConfigData();

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}



void CLoginDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CLoginDlg)
        DDX_Control(pDX, IDC_EDIT_USERNAME, m_Edit_UserName);
        DDX_Control(pDX, IDC_EDIT_PASSWORD, m_Edit_Password);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CLoginDlg, CDialog)
        //{{AFX_MSG_MAP(CLoginDlg)
        ON_BN_CLICKED(IDC_LOGIN, OnLogin)
        ON_BN_CLICKED(IDOK, OnOK)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLoginDlg message handlers

void CLoginDlg::OnLogin()
{
        unsigned short *temp = new unsigned short [LOGINLENGTH];
        unsigned short *temp1 = new unsigned short [PASSWORDLENGTH];
        int i,length;
        CString tempch;


        m_Edit_UserName.GetWindowText(temp,LOGINLENGTH);
        m_Edit_Password.GetWindowText(temp1,PASSWORDLENGTH);

        for(i=0;i<LOGINLENGTH;i++)
                cLoginName[i] = temp[i];
        for(i=0;i<PASSWORDLENGTH;i++)
                cPassword[i] = temp1[i];
        // Encryption of the login and pwd
        _strlwr( cLoginName );

        length = strlen(cLoginName);
        for (i = 0; i<length;i++)
                cLoginName[i] ^ = ENCRYPTKEY;

        length = strlen(cPassword);
        for (i = 0; i<length;i++)
                cPassword[i] ^= ENCRYPTKEY;


        if(!CheckLoginData())
        {
                MessageBox(_T("Try Again"), _T("Error"), MB_OK);
                m_Edit_UserName.SetWindowText(_T(""));
                m_Edit_Password.SetWindowText(_T(""));
                return;
        }

        CDialog::OnOK();
}

void CLoginDlg::OnOK()
{
        // TODO: Add extra cleanup here

        CDialog::OnCancel();
}

void CLoginDlg::OnCancel()
{
        // TODO: Add extra cleanup here

        CDialog::OnCancel();
}

bool CLoginDlg::CheckLoginData()
{
        FILE *File_Data;
        char *data = new char[120];
        char *UName = new char[50];
        char *Pwd = new char[30];
        char *FName = new char[30];
        int length,i;
```

```
		File_Data = fopen( LOCALLOGINSFILE, "r");
		if(File_Data == NULL)
			MessageBox(_T("Cannot find Login File"), _T("Error"),
MB_OK);


		if( strcmp(cLoginName, DEFAULTLOGINNAME) == 0 &&
strcmp(cPassword, DEFAULTPASSWORD) ==0 )
		{
			memcpy(cLoginName, "Admin", 30);
			return TRUE;
		}

		while(!feof(File_Data))
		{
			fgets(data, 120, File_Data);
			sscanf(data,"%s\t%s\t%s", UName, Pwd, FName);
			if( strcmp(cLoginName, UName)==0 && strcmp(cPassword,
Pwd)==0)
			{
				length = strlen(FName);
				for (i = 0; i<length;i++)
					cLoginName[i] = FName[i] ^ ENCRYPTKEY;
				cLoginName[i] = '\0';
	//			MessageBox(L"ok",0,MB_OK);
	//			memcpy (cLoginName, FName, 50);
	//			MessageBox(L"ok1",0,MB_OK);
				fclose (File_Data);
				return TRUE;
			}
		}
		fclose (File_Data);

		return FALSE;
	}
```

RssiTest.c:

```
#include "RssiTest.h"
//#include "Globals.h"


/**************** Globals********************************/
HCI_Handle ghDefaultHandle = NULL;
HCI_Handle ghConnHandle = NULL;
HCI_Handle ghScoHandle = NULL;
hci_link_type guScoLinkType = HCI_NO_LINK;
bt_inq_res gauInquiryResult[HCI_MAX_NUM_INQ_RES];
int giNumDevice = 0;
bt_device_context ghDeviceContext;
bd_addr guBdAddr;

int iDeviceIndex;
bd_addr Devices_BD_Addr[MAXNUMDEVICES];

uint16 giStatus;
HANDLE ghEvent;
uint8 bRssiInProgress = 0;


HANDLE ghWriteEvent;
HANDLE ghExitEvent;
HANDLE ghStopEvent ;

HANDLE ghWriteThread;
HANDLE ghRssiThread ;
app_hci_cmds guWriteCmd;
int giConnFlag;
int giProgressFlag;
MData MDummy;
MData* puMatchData=&MDummy;
uint32 aiEventArray[MAX_EVENTS];

char RemDummy[50];
char *RemoteName = RemDummy;
int iTotalDevices;


/*******************************************************************
Function Name  :HciTestInitialise
Function Type  :Internal
```

```
Description    :This is a main function which starts the
                              application.
Arguments      :
               Input  :

     Returns :none
*****************************************************************/

EXTERN int HciTestIntitialise(void )
{
   uint8 iEventCount;
   long iRetVal = 0;

   iEventCount = 0;
        aiEventArray[iEventCount++]  = HCI_DIS_COM_EVN;


        if((ghDeviceContext = BT_OpenDevice(BLUE_USB))==NULL)
   {
      printf("Unable to open device \n");
      exit(1);
   }
   /* Get a handle to HCI before executing any hci command */
        ghDefaultHandle = BT_HCI_GetHandle(ghDeviceContext,HCI_DEFAULT);
        if (!ghDefaultHandle)
        {
              printf("\n ");
              exit(1);
        }
        BT_HCI_RegisterCallBack(ghDefaultHandle,aiEventArray,(hci_callback)APP_C
onnectionCallback,iEventCount);


   return 1;
}

/****************************************************************
Function Name  :WriterThread
Function Type  :Internal
Description    :This is a Writer Thread's Function
Arguments      :
               Input  :LPVOID : ignored

     Returns :DWORD
*****************************************************************/
DWORD WriterThread(LPVOID pVoid)
```

```
{
        HciTestIntitialise();

        while(1)
        {
                WaitForSingleObject(ghWriteEvent,INFINITE);

                switch (guWriteCmd)
                {
                case HCI_INQUIRY:
                        APP_HciTestCommands(HCI_INQUIRY,puMatchData);
                        break;
                case HCI_CONNECT:
                        APP_HciTestCommands(HCI_CONNECT,NULL);
                        break;
                case HCI_DISCONNECT:
                        APP_HciTestCommands(HCI_DISCONNECT,NULL);
                        break;
                case HCI_READ_RSSI:
                        APP_HciTestCommands(HCI_READ_RSSI,puMatchData);
                        break;
                case HCI_READ_REMOTE_NAME:
                        APP_HciTestCommands(HCI_READ_REMOTE_NAME,
NULL);
                case EXIT:

                        BT_Close(0);
                        SetEvent(ghExitEvent);
                        return 0;
                }
        }
        return 0;
}




/*************************************************************************
***

        Function Name        : APP_HciTestCommands
        Description          : Function that issues various HCI commands
        Argument
```

```
            Input           :
hHciHandle,uEventId,pcData(EventData),iLength,pvMatchData
            Output          : None
        Global              : None.
        Error Condition     : -
        Return              : None
*************************************************************************
*****/

EXTERN void APP_HciTestCommands(int iCmdType,void* pvMatchData)
{
   hci_callback_info uCallBackInfo ;
   switch(iCmdType)
   {
   case HCI_INQUIRY:
      {
         lower_bd_addr uLap;
         uLap.iData[0] = GIAC >> SHIFT_16;
         uLap.iData[1] = GIAC >> SHIFT_8  & HCI_LOWER_BYTE_MASK;
         uLap.iData[2] = GIAC & HCI_LOWER_BYTE_MASK;

         uCallBackInfo.pfCallBack = (hci_callback)APP_InquiryCallback;
         uCallBackInfo.pvMatchData = pvMatchData;
         printf("\n Issuing Inquiry Command ...");
         giNumDevice = 0;

                        // Check giStatus before waiting on a event.
                        giStatus = BT_HCI_Inquiry(ghDefaultHandle,
                                   uLap,
                                   INQUIRY_DURATION,
                                   HCI_INQUIRY_MAX_RESPONSES,
                                   uCallBackInfo) ;
         if(giStatus != HCI_SUCCESS)
                        {
            SetEvent(ghEvent);
         }

         break;
      }
   case HCI_CONNECT:
      {
         printf("\n Issuing Connect Command ...");
         uCallBackInfo.pfCallBack = (hci_callback)APP_ConnectionCallback;
         uCallBackInfo.pvMatchData = NULL ;

         ghConnHandle = BT_HCI_GetHandle(ghDeviceContext,0);
        giStatus = BT_HCI_CreateConnection(ghConnHandle,
                                guBdAddr,
                                uCallBackInfo);
        if (giStatus != HCI_SUCCESS)
                        {
           SetEvent(ghEvent);
        }
                        break;
    }
   case HCI_DISCONNECT:
    {
        uint8 iReason = HCI_USER_TERMINATE;
        uCallBackInfo.pfCallBack = (hci_callback)APP_ConnectionCallback;
        uCallBackInfo.pvMatchData = NULL ;

        giStatus = BT_HCI_Disconnect(ghConnHandle,
                                iReason,
                                uCallBackInfo);
        if (giStatus != HCI_SUCCESS)
                        {
           SetEvent(ghEvent);
        }
                        break;
    }
        case HCI_READ_RSSI:
                {
        printf("\n Issuing rssi Command ...");
        uCallBackInfo.pfCallBack = (hci_callback)APP_ConnectionCallback;
        uCallBackInfo.pvMatchData = pvMatchData ;
        giStatus = BT_HCI_ReadRssi(ghConnHandle,
                                uCallBackInfo);
        if (giStatus != HCI_SUCCESS)
                        {
                                ghStopFlag = TRUE;
           SetEvent(ghEvent);
        }
                        break;
                }
        case HCI_READ_REMOTE_NAME:
                {
                        MessageBox(NULL, T("asasas"),NULL,MB_OK);
                        uCallBackInfo.pfCallBack =
(hci_callback)APP_RemoteNameCallback;
                        uCallBackInfo.pvMatchData = NULL;
                        giStatus   BT_HCI_RemoteNameRequest(ghConnHandle,
                                gauInquiryResult[iDeviceIndex].uBdAddr,
                                gauInquiryResult[iDeviceIndex].iPageScanRepModes,
                                gauInquiryResult[iDeviceIndex].iPageScanModes,
                                gauInquiryResult[iDeviceIndex].iClkOffset,
                                uCallBackInfo);
        if(giStatus == HCI_FAILURE)
                           {
                                MessageBox(NULL, _T("Stopped"),NULL,MB_OK);
                                ghStopFlag = TRUE;
          SetEvent(ghEvent);
        }
                           break;
                  }
        }

}



/*********************************************************************************
***

        Function Name           : APP_InquiryCallback
        Description             : Callback function registered with HCI for inquiry
        Argument
              Input             :
hHciHandle,uEventId,pcData(EventData),iLength,pvMatchData
              Output            : None
        Global                  : None.
        Error Condition         : -
        Return                  : None
**********************************************************************************
*****/
void APP_InquiryCallback (   HCI_Handle hHciHandle,
                      uint32 uEventId,
                      uint8 *pcData,
                      uint32 iLength ,
                                          void *pvMatchData)
{
   int i;
        static int iFlag=0;
        static count;
   if (uEventId == HCI_INQ_RES_EVN)
   {
     hci_inq_res *puInquiryResult;
     puInquiryResult = (hci_inq_res *)pcData;
```

```
                iFlag=1;
                // for each inquiry response
        for (i=0 ; i<puInquiryResult->iNumResponses ; i++)
        {
          /*  Store the inquiry result details */
          gauInquiryResult[giNumDevice] = puInquiryResult->auBtInqRes[i];
          giNumDevice++;

                        // added
                        memcpy((((MData*)pvMatchData)-
>uiBuf+BD_ADDR_SIZE*count),puInquiryResult-
>auBtInqRes[i].uBdAddr.iData,BD_ADDR_SIZE);
                        ((MData*)pvMatchData)->uiNumber=++count;
        }
      }
      else if (uEventId == HCI_INQ_CMP_EVN)
      {
        hci_inq_cmp *puInquiryComplete;
        puInquiryComplete = (hci_inq_cmp *)pcData;
        giStatus = puInquiryComplete->iStatus;

        /*  Go back to main */
                if (!iFlag)
                {
                        ((MData*)pvMatchData)->uiNumber=0;
                }
                count=0;
                iFlag=0;

                giProgressFlag=0;
        SetEvent(ghEvent);
      }
      else if (uEventId =  HCI_COM_STA_EVN)
      {
        hci_cmd_status_event *puInqStatus;
        puInqStatus = (hci_cmd_status_event *)pcData;
        giStatus = puInqStatus->iStatus;
        if (giStatus != HCI_SUCCESS)
        {
          giNumDevice = 0;
          SetEvent(ghEvent);
        }
      }
          else if (uEventId == HCI_TIME_OUT_EVN)
          {
                //err
        }
}




/*****************************************************************
***

        Function Name           : APP_ConnectionCallback
        Description             : Callback function registered with HCI for connection
        Argument
                Input           :
hHciHandle,uEventId,pcData(EventData),iLength,pvMatchData
                Output          : None
        Global                  : None.
        Error Condition         : -
        Return                  : None
*****************************************************************
*****/

void APP_ConnectionCallback(HCI_Handle hConn,
                uint32 uEventCode,
                void *pcData,
                                                uint32 iLength,
                void *pvMatchData)
{
  long iRetVal  =0;
        switch(uEventCode)
        {
  case HCI_COM_COM_EVN:
                {
                        hci_cmd_comp_event *puEvent;
                        puEvent = (hci_cmd_comp_event *)pcData;
                        if (puEvent->iCommandOpcode==HCI_READ_RSSI_CMD)
                        {
                                memcpy(((MData*)pvMatchData)->uiBuf,puEvent-
>iReturnParams,60);
                        }
        SetEvent(ghEvent);
        break;
    }
  case HCI_COM_STA_EVN:
     {/* Command Status Event     */
```

```
   hci_cmd_status_event *puInqStatus;
   puInqStatus = (hci_cmd_status_event *)pcData;
   giStatus = puInqStatus->iStatus;
   if (giStatus != HCI_SUCCESS)
                   {
      SetEvent(ghEvent);
   }
   break;
}
   case HCI_CON_COM_EVN:
          {
                   hci_con_complete event *uConnInfo;
    uint16 iPacketType   0x0400;
                   uConnInfo = (hci_con_complete_event *)pcData;
   printf("\n HCI_CON_COM_EVN event");
   if (uConnInfo->iStatus ==  HCI_SUCCESS)
   {
      printf("\n Connection is complete ");
      if (uConnInfo->uLinkType == HCI_ACL)
      {
         ghConnHandle = hConn;
      }
      else
      {
         ghScoHandle = hConn;
         guScoLinkType = uConnInfo->uLinkType;
      }
                  // added this for RSSI
                  giConnFlag  1;

   }
                  else
                  {
                         printf("\n Connection is incomplete ");
                         // added for RSSI
                         giConnFlag=0;

                  }
                  SetEvent(ghEvent);
                  break;
          }
   case HCI_DIS_COM_EVN:
          {
                  hci_discon_complete_event *uDisconnInfo;
                  uDisconnInfo = (hci_discon_complete_event *)pcData;
```

```
                        printf(" Disconnection occurred ");
                        // added for RSSI
                        giConnFlag=0;
                        SetEvent(ghEvent);
                        break;
                }
 default:
                {
                        printf("Invalid Event");
                }
        }
        return;
}


/***********************************************************************
***

        Function Name           : APP_RemoteNameCallback
        Description             : Callback function registered with HCI for Remote name
req
        Argument
                Input           :
hHciHandle,uEventId,pcData(EventData),iLength,pvMatchData
                Output          : None
        Global                  : None.
        Error Condition         : -
        Return                  : None
************************************************************************
*****/


void APP_RemoteNameCallback(HCI_Handle hConn,
                uint32 uEventCode,
                void *pcData,
                                                uint32 iLength,
                void *pvMatchData)
{
        MessageBox(NULL,_T("gone"),NULL,MB_OK);
        if ( uEventCode == HCI_REM_NAM_REQ_EVN )
        {
                hci_remote_name_req_event *puRemoteNameEvt;
                puRemoteNameEvt   (hci_remote_name_req_event *)pcData;
                memcpy(RemoteName,
                        puRemoteNameEvt->uName.acData,
                    sizeof(puRemoteNameEvt->uName.acData) );
        }
        SetEvent(ghEvent);
}
```

```
/*********************************************************************************
***
        Function Name           : Util_LogBdAddr
        Description             : Logs the given bdaddr into the specified file
        Argument
                Input           : fPtr,uBdAddr
                Output          : None
        Global                  : None.
        Error Condition         : -
        Return                  : None
**********************************************************************************
*****/
void Util_LogBdAddr(FILE *fPtr,bd_addr uBdAddr)
{
        int i;
        for(i=0 ; i<BD_ADDR_SIZE ; i++)
        {
                printf("%x ",uBdAddr.iData[i]);
        }
}
```

ui.cpp:

```
// ui.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "ui.h"
//#include "Globals.h"
#include "uiDlg.h"
#include "RssiTest.h"
#include "CLoginDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////////////////////////////////
// CUiApp

BEGIN_MESSAGE_MAP(CUiApp, CWinApp)
	//{{AFX_MSG_MAP(CUiApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUiApp construction

CUiApp::CUiApp()
	: CWinApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CUiApp object

CUiApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CUiApp initialization
```

```
BOOL CUiApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.
   cLoginName = new char[LOGINLENGTH];
        cPassword = new char[PASSWORDLENGTH];
        cLoginName = "Moses";

        int nResponse;

        CLoginDlg LoginDlg;
        nResponse = LoginDlg.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                delete cLoginName;
                delete cPassword;
                exit(0);
        }

        CUiDlg dlg;
        m_pMainWnd = &dlg;
        nResponse = dlg.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
                if(bRssiInProgress)
                {
                        //SetEvent(ghStopEvent);
                        ghStopFlag=true;
                        Sleep(2000);
                }
                guWriteCmd=EXIT;
                iAppFlag=1;
                SetEvent(ghWriteEvent);
                // waiting for the exit event to complete
                WaitForSingleObject(ghExitEvent,INFINITE);
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }

        // Since the dialog has been closed, return FALSE so that we exit the
        //  application, rather than start the application's message pump.
        return FALSE;
}
```

uiDlg.cpp:

```
// uiDlg.cpp : implementation file
//

#include "stdafx.h"
#include "ui.h"
#include "uiDlg.h"
#include "RssiTest.h"
//#include "Globals.h"
#include <winsock.h>
#include "ServDisc.h"
#include <winbase.h>
#include <Winreg.h>

//extern bt_device_context ghDeviceContext;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

int iAppFlag  0;
BOOL ghStopFlag;
char *cLoginName;
char *cPassword;

//HANDLE ghStopCompleteEvent;

/////////////////////////////////////////////////////////////////////////////
// CUiDlg dialog

CUiDlg::CUiDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CUiDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CUiDlg)
	m_BDAddr = _T("");
	m_RSSIval = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

//	m_Button_NewMsgs.EnableWindow(FALSE);

	bConnPresence = FALSE;
```

```
bAutoOn = FALSE;
bFileSelect = FALSE;
bProgramBegin = TRUE;
bFTPPresence = FALSE;

ConnectionHandle = new HANDLE;
ConnectionInfo = new CONNMGR_CONNECTIONINFO;
DestinationInfo = new CONNMGR_DESTINATION_INFO;
bFileTransferProgress = FALSE;

cDevices_BD_Addr[0] = new char[14];
cDevices_BD_Addr[1] = new char[14];
cDevices_BD_Addr[2] = new char[14];
cDevices_BD_Addr[3] = new char[14];
cDevices_BD_Addr[4] = new char[14];

cRemoteNames[0] = new char[20];
cRemoteNames[1] = new char[20];
cRemoteNames[2] = new char[20];
cRemoteNames[3] = new char[20];
cRemoteNames[4] = new char[20];

NumMessages = 0;
MessagesDisplayed = 0;
COM_File = NULL;

for ( int i = 0; i<MAXNUMMESSAGES; i++)
        Messages[i] = new char[100];

*ConnectionHandle = ConnMgrApiReadyEvent();

iDeviceIndex = -1;

FILE *File_Config;

File_Config = fopen(CONFIGFILE,"r");

if(File_Config == NULL)
{
        MessageBox(_T("Config File not found"),_T("Error"),MB_OK);
        exit(0);
}

char temp[20];

fscanf(File_Config,"%s",temp);
```

```
        Server_IP_Address = temp;

        fscanf(File_Config,"%s",temp);
        FTP_UserName = temp;

        fscanf(File_Config,"%s",temp);
        FTP_Password = temp;
        int length;

        fscanf(File_Config,"%s",temp);
        length = strlen(temp);
        if (temp[ length-1 ]=='/') temp[ length - 1 ] = '\0';
        MainRemoteFolder = temp;

        fscanf(File_Config,"%s",temp);
        DestinationConnection = temp;

        fclose (File_Config);

//      BT_Sdap_Handle = NULL;
//      Start_BTCMN(ghDeviceContext);
//      CreateServiceClassID();

/*      CString ch2;
        ch2 = "lynnmg: How does Bluetooth work?";
        m_List_NewMsgs.SetWindowText(ch2);
*/
}

void CUiDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CUiDlg)
        DDX_Control(pDX, IDC_RSSI_LIST, m_RSSI_List);
        DDX_Control(pDX, IDC_INQ_LIST, m_InqList);
        DDX_Control(pDX, IDC_LIST_MESSAGES, m_List_NewMsgs);
        DDX_Control(pDX, IDC_NEWMSGS_BUTTON, m_Button_NewMsgs);
        DDX_Control(pDX, IDC_EDIT_FILENAME, m_Edit_FileName);
        DDX_Control(pDX, IDC_STATUS, m_Edit_Status);
        DDX_Control(pDX, IDC_PROGRESS, m_InqProgress);
        DDX_LBString(pDX, IDC_INQ_LIST, m_BDAddr);
        DDX_Text(pDX, IDC_RSSI_EDIT, m_RSSIval);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CUiDlg, CDialog)
```

```
        //{{AFX_MSG_MAP(CUiDlg)
        ON_BN_CLICKED(IDC_INQ_BUTTON, OnInqButton)
        ON_BN_CLICKED(IDC_RSSI_BUTTON, OnRssiButton)
        ON_LBN_SELCHANGE(IDC_INQ_LIST, OnSelchangeInqList)
        ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
        ON_WM_CLOSE()
        ON_BN_CLICKED(IDC_CONN_BUTTON, OnConnButton)
        ON_BN_CLICKED(IDC_DISCONN_BUTTON, OnDisconnButton)
        ON_BN_CLICKED(IDC_AUTO_BUTTON, OnAutoButton)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_FILESEND_BUTTON, OnFilesendButton)
        ON_BN_CLICKED(IDC_BROWSE_BUTTON, OnBrowseButton)
        ON_LBN_SELCHANGE(IDC_RSSI_LIST, OnSelchangeRssiList)
        ON_BN_CLICKED(IDC_NEWMSGS_BUTTON, OnNewmsgsButton)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUiDlg message handlers

BOOL CUiDlg::OnInitDialog()
{

        DWORD dwThreadID;

        CDialog::OnInitDialog();

// creartion of the events
        ghEvent = CreateEvent(NULL,FALSE,FALSE,NULL);
        ghWriteEvent = CreateEvent(NULL,FALSE,FALSE,NULL);
        ghExitEvent = CreateEvent(NULL,FALSE,FALSE,NULL);
        ghStopEvent = CreateEvent(NULL,FALSE,FALSE,NULL);

        ghStopFlag=false;

// initialisation of the globals
        giConnFlag=0;
        giProgressFlag=1;
        m_BDAddr="";
        m_InqProgress.SetRange(0,130);

// creation of the thread
        if (!iAppFlag)
        {
   ghWriteThread = CreateThread (NULL, 0, WriterThread, NULL, 0,
                        &dwThreadID);
        if(!ghWriteThread)
   {
    // Could not create the read thread.
    DWORD dwLastError=GetLastError();
        }

// wait for initialisation to get complete
//      WaitForSingleObject(ghEvent,INFINITE);
        }

        // Set the icon for this dialog.  The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                     // Set big icon
        SetIcon(m_hIcon, FALSE);            // Set small icon
        CenterWindow(GetDesktopWindow());        // center to the hpc screen

        m_iAvgRssiVal = RSSI_INVALID_VALUE;

        // TODO: Add extra initialization here

        if( !DestinationNetwork() )
        {
                MessageBox(_T("Connection Manager Destination not found"),
 T("Error"),MB_OK);
                exit(0);
        }

//      m_Edit_Messages.SetWindowText(_T("this is a message new
messagekkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
kkkkkkkkkkkkkkkkkkkkkkkkkkkkk"));//,NULL,MB_OK);

        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_DISCONN_BUTTON)->EnableWindow(FALSE);
        m_Button_NewMsgs.EnableWindow(FALSE);

        return TRUE; // return TRUE  unless you set the focus to a control
}

/////////////////////////////////////////////////////////////////////////////

void CUiDlg::OnInqButton()
{
        // TODO: Add your control notification handler code here
        uint8 szTemp[7];
        int i=0,iProgress=0;
```

```
CWnd* pcwListBox = this->GetDlgItem(IDC_INQ_LIST );

      m_Edit Status.SetWindowText(_T("Busy!"));

      m_RSSI_List.ResetContent();

      iDeviceIndex = 0;

      if(bRssiInProgress)
      {
               return ;
      }

      // resetting intial values
      giProgressFlag=1;
      ListBox_ResetContent(pcwListBox->m_hWnd);
      m_RSSIval=_T("");
      CEdit * pcwEditBox = (CEdit *)this->GetDlgItem(IDC_RSSI_EDIT );
      pcwEditBox->SetWindowText(m_RSSIval);
      m_BDAddr= "";

      // setting the variables for inquiry &
      // unblocking the write thread
      guWriteCmd HCI_INQUIRY;
      SetEvent(ghWriteEvent);

      // displaying the progress bar with the time =13 sec (~1.28*10)
      while (giProgressFlag)
      {
               Sleep(500);
               iProgress+=5;
               m_InqProgress.SetPos(iProgress);
      }
      WaitForSingleObject(ghEvent,INFINITE);

      // checking for the number of values returned
      if (!(puMatchData->uiNumber))
      {
               MessageBox(L"No Devices Within Range",
                              L"Warning",
                              MB_OK|MB_ICONWARNING);
               m_InqProgress.SetPos(0);
               return;
      }

      // displaying the results
```

```
        iTotalDevices = puMatchData->uiNumber;
        for(i=0;i<iTotalDevices;i++)
        {
                //GetRemoteName(i);
                iDeviceIndex = i;
                memcpy(szTemp,(puMatchData->uiBuf+ BD_ADDR_SIZE*i),
BD_ADDR_SIZE);
                AddToListBox(szTemp);
                m_RSSI_List.InsertString(iDeviceIndex, _T("NA"));
        }

//      if(bReceivedMapping)
        AssignRemoteNames();

        iDeviceIndex = -1;

        // resetting the progress bar
        m_InqProgress.SetPos(0);
        if(!bAutoOn)
        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(TRUE);
        if(!bAutoOn) m_Edit_Status.SetWindowText(_T("Done!"));

}


/////////////////////////////////////////////////////////////

void CUiDlg::OnRssiButton()
{
        DWORD dwThreadID ;
        char cTempVal[50];
        // TODO: Add your control notification handler code here
        // intial check for the device
   CWnd* pcwListBox = this->GetDlgItem(IDC_INQ_LIST );
        CString TempRssi;
//   CWnd* pcwListBox = this->GetDlgItem(IDC_INQ_LIST );


        int nIndex = ListBox_GetCurSel(pcwListBox->m_hWnd);

        if(iDeviceIndex == -1)
        {
               MessageBox(L"Please Select the Device",
                               L"Error",
                               MB_OK|MB_ICONERROR);
        return ;

}

/* If RSSI is pressed twice , dont act */
if (bRssiInProgress == 1)
{
        ghStopFlag=true;
        GetDlgItem(IDC_RSSI_BUTTON)->SetWindowText(TEXT("RSSI"));
        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(TRUE);
        m_InqProgress.SetPos(0);

        sprintf(cTempVal,"%d",this->m_iAvgRssiVal);
        TempRssi = cTempVal;
        sprintf(cTempVal,"%d", this->m_iAvgRssiVal);
        m_RSSI_List.DeleteString(nIndex);
        m_RSSI_List.InsertString(nIndex, TempRssi);
        m_Edit_Status.SetWindowText(_T("Done!"));
        m_RSSIval = cTempVal;

        // displaying the rssi value
        CEdit * pcwEditBox = (CEdit *)GetDlgItem(IDC_RSSI_EDIT);
        pcwEditBox->SetWindowText(m_RSSIval);

        GetDlgItem(IDC_RSSI_BUTTON)->SetWindowText(TEXT("RSSI"));
        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(TRUE);
        GetDlgItem(IDC_INQ_LIST)->EnableWindow(TRUE);
        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(TRUE);
        GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(TRUE);


        return ;
}
else
{
        this->m_iAvgRssiVal = 0;
        ghStopFlag=false;
        m_RSSIval = "";
        CEdit * pcwEditBox = (CEdit *)GetDlgItem(IDC_RSSI_EDIT );
        pcwEditBox->SetWindowText(m_RSSIval);

        GetDlgItem(IDC_RSSI_BUTTON)->SetWindowText(TEXT("Stop"));
        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_INQ_LIST)->EnableWindow(FALSE);
        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(FALSE);
```

```
			ghRssiThread = CreateThread (NULL, 0, RssiThread, (LPVOID)this, 0,
				&dwThreadID);
			m_Edit_Status.SetWindowText(_T("Busy!"));
		}

}



///////////////////////////////////////////////////

void CUiDlg::AddToListBox(uint8* szBuf)
{
		wchar_t szBufW[14];
		uint8 uiTemp =0x00;
		int8* acTemp,*acTempBD[BD_ADDR_SIZE];
		CString ch;

  //get the control window
  //set text to show in control
		for(int i=0;i<6;i++)
		{
			uiTemp=(szBuf[i] & 0xf0)>>4;
			_itow(uiTemp,&szBufW[2*i],16);
			uiTemp= szBuf[i] & 0x0f;
			_itow(uiTemp,&szBufW[2*i+1],16);
		}
		szBufW[12]='\0';

		for(i=0;i<13;i++)
			cDevices_BD_Addr[iDeviceIndex][i] = szBufW[i];

//		ch = cDevices_BD_Addr[iDeviceIndex];
//		MessageBox(ch,NULL,MB_OK);

		for (i 0;i<BD_ADDR_SIZE;i++)
		{
			acTemp=(int8*)malloc(3*sizeof(int8));
			*acTemp=(int8)szBufW[2*i];
			*(acTemp+1)=(int8)szBufW[2*i+1];
			acTemp[2]='\0';
			acTempBD[i]=acTemp;
		}
		UTILS_ExtractBdAddr(BD_ADDR SIZE,&acTempBD[0],&guBdAddr);

		Devices_BD_Addr[iDeviceIndex] = guBdAddr;
```

```
   //ListBox_AddString(pcwListBox->m_hWnd, szBufW);
        ch = szBufW;
        m_InqList.InsertString(iDeviceIndex, ch);
}



////////////////////////////////////////////////////////////

void CUiDlg::OnSelchangeInqList()
{
        // TODO: Add your control notification handler code here
        TCHAR szBufW[14];
        int nIndex=0;
        *szBufW='\0';
        CString c;

        if(COM_File != NULL)
                CloseHandle(COM_File);

        // TODO: Add your control notification handler code here
        // copying the selected value to m_BDAddr
   CWnd* pcwListBox = this->GetDlgItem(IDC_INQ_LIST );
        nIndex = ListBox_GetCurSel(pcwListBox->m_hWnd);
        iDeviceIndex = nIndex;
//      ListBox_GetText(pcwListBox->m_hWnd, nIndex,(LPTSTR)szBufW );
//      m_BDAddr = szBufW;

        // resetting the RSSI display
        m_RSSIval=_T("");
        CEdit * pcwEditBox = (CEdit *)this->GetDlgItem(IDC_RSSI_EDIT );
        pcwEditBox->SetWindowText(m_RSSIval);
        m_RSSI_List.SetCurSel(nIndex);


/*      Start_BTCMN();
        SdapOpenConnection(ghDeviceContext, Devices_BD_Addr[iDeviceIndex]);
        MessageBox(L"Open", NULL, MB_OK);

//      CreateServiceClassID();
//      MessageBox(L"Create", NULL, MB_OK);

        DiscoverService();
        MessageBox(L"Discover", NULL, MB_OK);
```

```
//      GetDLC();
//      MessageBox(L"DLC", NULL, MB_OK);

        SdapCloseConnection();
        MessageBox(L"Close", NULL, MB_OK);

        Stop_BTCMN();
*/
//      SetRegistryValues();

//      MessageBox(L"Start",NULL,MB_OK);
/*      if( !CreateCOMPort() )
                MessageBox(L"Failure",NULL,MB_OK);
*/
//      MessageBox(L"End",NULL,MB_OK);
        //CloseHandle(COM_File);
}


void CUiDlg::OnExitButton()
{
        // TODO: Add your control notification handler code here

        // setting the variables for exit &
        // unblocking the write thread
        DWORD Conn_Status;
        ConnMgrConnectionStatus(*ConnectionHandle, &Conn_Status);
        if(Conn_Status != CONNMGR_STATUS_CONNECTED)
        {
                bConnPresence = FALSE;
//              bFTPPresence = FALSE;
        }

        if(bAutoOn) KillTimer(CONNREFRESHTIMERID);

        if(bConnPresence)
                OnDisconnButton();

        if (bRssiInProgress)
        {
                ghStopFlag = true;
                Sleep(2000);
        }
        guWriteCmd=EXIT;
        iAppFlag=1;
        SetEvent(ghWriteEvent);
        // waiting for the exit event to complete
        WaitForSingleObject(ghExitEvent,INFINITE);
        ExitThread(0);
        CDialog::OnClose();
}


void CUiDlg::OnClose()
{
        // TODO: Add your message handler code here and/or call default

        if(bAutoOn) KillTimer(CONNREFRESHTIMERID);

        if(bConnPresence)
                OnDisconnButton();

        while(bRssiInProgress)
        {
                SetEvent(ghStopEvent);
        }
        guWriteCmd=EXIT;
        iAppFlag=1;
        SetEvent(ghWriteEvent);
        // waiting for the exit event to complete
        WaitForSingleObject(ghExitEvent,INFINITE);

        CDialog::OnClose();
}



DWORD RssiThread(LPVOID pvContext)
{
        int8* acTemp,*acTempBD[BD_ADDR_SIZE];
        char cTempVal[30],cTempNum[4];
        int i,iProgress=0;
        DWORD  iStatus = STATUS_TIMEOUT ;
        CUiDlg *puUI   (CUiDlg *)pvContext ;


        // converting the selected text to a valid BD_ADDR
/*      for (i=0;i<BD_ADDR_SIZE;i++)
        {
                acTemp-(int8*)malloc(3*sizeof(int8));
                if (!acTemp)
                        return 1;
        *acTemp=(int8)(puUI->m_BDAddr)[2*i];
        *(acTemp+1)= (int8)(puUI->m_BDAddr)[2*i+1];
        acTemp[2]='\0';
        acTempBD[i]=acTemp;
    }
    UTILS_ExtractBdAddr(BD_ADDR_SIZE,&acTempBD[0],&guBdAddr);
*/
    guBdAddr = Devices_BD_Addr[iDeviceIndex];

    // setting the variables for connect &
    // unblocking the write thread
    guWriteCmd=HCI_CONNECT;
    SetEvent(ghWriteEvent);

    puUI->GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(FALSE);


    // displaying the progress bar with the time ~ 1.3 * 5  sec
    for (i=0;i<14;i++)
    {
        Sleep(100*5);
        iProgress+=5;
        puUI->m_InqProgress.SetPos(iProgress);

    }

    WaitForSingleObject(ghEvent,INFINITE);

    puUI->GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(TRUE);


    //checking for the connection
    if (! giConnFlag)
    {
        puUI->MessageBox(L"Unable to Connect to Remote Device",
                        L"Error",
                        MB_OK|MB_ICONERROR);
        puUI->m_InqProgress.SetPos(0);
        bRssiInProgress = 0;

    puUI->GetDlgItem(IDC_RSSI_BUTTON)->SetWindowText(TEXT("RSSI"));
    puUI->GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(TRUE);
    puUI->GetDlgItem(IDC_INQ_LIST)->EnableWindow(TRUE);

        ExitThread(1);
        return 1;
```

```
}


while (!ghStopFlag)
{

        bRssiInProgress = 1;
        // setting the variables for read rssi &
        // unblocking the write thread
        guWriteCmd=HCI_READ_RSSI;
        SetEvent(ghWriteEvent);
        // displaying the progress bar with the time ~ 2.6  sec
        for (i=0,iProgress=0;i<26;i++)
        {
                Sleep(100);
                iProgress+=5;
                puUI->m_InqProgress.SetPos(iProgress);
                if (ghStopFlag)
                {
                        iProgress=0;
                        puUI->m_InqProgress.SetPos(iProgress);
                        break;
                }

        }
        WaitForSingleObject(ghEvent,INFINITE);
        if (puMatchData->uiBuf[3] & 0x80)
        {
                puUI->m_iAvgRssiVal += (int8)-((~puMatchData->uiBuf[3])+1) ;
        }
        else
        {
                puUI->m_iAvgRssiVal += (int8)(puMatchData->uiBuf[3]) ;
        }

        if ((puUI->m_RSSIval.Compare(_T(""))))
        {
                puUI->m_iAvgRssiVal /= 2 ;
        }

        if (!(puMatchData->uiBuf[3] | 0x00))
        {
                //strcpy(cTempVal,"Normal RSSI strength = ");
                _itoa(puMatchData->uiBuf[3],cTempNum,10);
                strcpy(cTempVal,cTempNum);
        }
```

```
		else if ((puMatchData->uiBuf[3] & 0x80))
		{
			int iTemp=0;
			strcpy(cTempVal,"-");
			iTemp=puMatchData->uiBuf[3];
			iTemp=~iTemp;
			iTemp&=0x00ff;
			iTemp+=1;
			_itoa(iTemp,cTempNum,10);
			strcat(cTempVal,cTempNum);
		}
		else
		{
			//strcpy(cTempVal,"Above Normal = ");
			_itoa(puMatchData->uiBuf[3],cTempNum,10);
			strcpy(cTempVal,cTempNum);
		}

		// assigning the value to m_RSSIval
		puUI->m_RSSIval=cTempVal;
		// displaying the rssi value
		if(!ghStopFlag)
		{
			CEdit * pcwEditBox = (CEdit *)puUI-
>GetDlgItem(IDC_RSSI_EDIT );
			pcwEditBox->SetWindowText(puUI->m_RSSIval);
			//sprintf(chInqTemp,"%x - %s",InqItem, cTempNum);
//			strcpy(chInqTemp1,chInqTemp);
//			strcat(chInqTemp1,cTempNum);
//			CInqItem = chInqTemp1;
//			puUI->m_InqList.DeleteString( index );
//			puUI->m_InqList.InsertString( index, CInqItem );
		}

		// resetting the progress bar
		puUI->m_InqProgress.SetPos(0);
	}



	// setting the variables for disconnect &
	// unblocking the write thread
	guWriteCmd=HCI_DISCONNECT;
	SetEvent(ghWriteEvent);
	WaitForSingleObject(ghEvent,INFINITE);
	bRssiInProgress= 0;
```

```
        // freeing the memory
        for (i=0;i<BD_ADDR_SIZE;i++)
        {
                if (acTempBD[i])
                        free(acTempBD[i]);
        }
        ExitThread(0);
        return 0 ;
}


void CUiDlg::OnConnButton()
{
        CString uErrorMsg = "Error in Connecting";
        CString uSuccessMsg = "Success in Connecting";
        DWORD *Conn_Status = new unsigned long;

        m_Edit_Status.SetWindowText(_T("Busy!"));

        int i,iProgress=0;
        DWORD  iStatus = STATUS_TIMEOUT;
        // converting the selected text to a valid BD_ADDR

//      if ( !CreateCOMPort() )
//              MessageBox(L"Error",NULL,MB_OK);

        if (iDeviceIndex == -1)
        {
                MessageBox(L"Please Select the Device",
                                L"Error",
                                MB_OK|MB_ICONERROR);
                return ;
        }

        GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(FALSE);
//      GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(FALSE);

        guBdAddr = Devices_BD_Addr[iDeviceIndex];

//      SetTimer( CHECKMESSAGESTIMERID, CHECKMESSAGESTIMEOUT,
NULL);

        bConnPresence = TRUE;
```

```
	SetRegistryValues();

	ConnMgrEstablishConnectionSync( ConnectionInfo, ConnectionHandle,
CONNTIMEOUT, Conn_Status);

	bConnPresence = TRUE;

	PrepareLocation();

/*	if( !FTPOpen() )
		MessageBox( _T("Error in Opening FTP"), _T("FTP"), MB_OK);
	else if ( !FTPSendFile() )
		MessageBox( _T("Error in Sending File"), _T("FTP"), MB_OK);*/
//	else ( FTPClose() );

//	}

//	SetTimer( CHECKMESSAGESTIMERID, CHECKMESSAGESTIMEOUT,
NULL);

	GetDlgItem(IDC_DISCONN_BUTTON)->EnableWindow(TRUE);
	m_Edit_Status.SetWindowText( T("Done!"));
}



void CUiDlg::OnDisconnButton()
{
	CString uErrorMsg = "Error in Disconnecting";
	CString uSuccessMsg = "Success in Disconnecting";
	char *ch = new char[12];
	CString c;

	m_Edit_Status.SetWindowText(_T("Busy!"));
	GetDlgItem(IDC DISCONN_BUTTON)->EnableWindow(FALSE);
//	GetDlgItem(IDC AUTO_BUTTON)->EnableWindow(FALSE);


	while(bRssiInProgress)
	{
		SetEvent(ghStopEvent);
	}

//	guWriteCmd =HCI_DISCONNECT;
//	SetEvent(ghWriteEvent);
//	WaitForSingleObject(ghEvent,INFINITE);
```

```
        while(bFileTransferProgress);

        KillTimer( CHECKMESSAGESTIMERID );

        PrepareLocation();
//      if( !FTPOpen() )
//              MessageBox( _T("Error in Opening FTP"), _T("FTP"), MB_OK);
//      else
                if( !FTPDeleteFile() )
                MessageBox( _T("Error in Sending File"), _T("FTP"), MB_OK);
        else ( FTPClose() );

        ConnMgrReleaseConnection(*ConnectionHandle, FALSE);
        Sleep(6000);

        CloseHandle(ConnectionHandle);

        bConnPresence = FALSE;

        if(bAutoOn)
        {
                bAutoOn = FALSE;
                KillTimer(CONNREFRESHTIMERID);
        }

        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(TRUE);
        GetDlgItem(IDC_DISCONN_BUTTON)->EnableWindow(FALSE);
        GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(TRUE);
        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(TRUE);
        GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(TRUE);

        m_Edit_Status.SetWindowText(_T("Done!"));

/*      CString c;
        Start_BTCMN(ghDeviceContext);
        MessageBox(L"Start",NULL,MB_OK);
        SdapOpenConnection(ghDeviceContext, Devices_BD_Addr[iDeviceIndex]);
        MessageBox(L"Open",NULL,MB_OK);
        CreateServiceClassID();
        MessageBox(L"Create",NULL,MB_OK);
        if(!DiscoverService())
        {
                c = szErrorString;
                MessageBox(c,NULL,MB_OK);
        }
```

```
	else
		MessageBox(L"Discover",NULL,MB_OK);
	SdapCloseConnection();
	MessageBox(L"Close",NULL,MB_OK);
	Stop_BTCMN();
*/
	*ConnectionHandle = ConnMgrApiReadyEvent();

}

void CUiDlg::OnAutoButton()
{

/*	for(i=0;i<(puMatchData->uiNumber);i++)
	{
		memcpy(szTemp,(puMatchData->uiBuf+
BD_ADDR_SIZE*i),BD_ADDR_SIZE);
		AddToListBox(szTemp);
	}
*/
	bAutoOn = TRUE;

	SetTimer( CONNREFRESHTIMERID, CONNREFRESHTIMEOUT, NULL);
//	SetTimer( CHECKMESSAGESTIMERID, CHECKMESSAGESTIMEOUT,
NULL);

	GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(FALSE);
	GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(FALSE);

	if ( !bFileTransferProgress ) AutoConnection();
}


void CUiDlg::OnTimer(UINT nIDEvent)
{
	DWORD Conn_Status;
	// TODO: Add your message handler code here and/or call default
	ConnMgrConnectionStatus(*ConnectionHandle, &Conn_Status);
	if(Conn_Status != CONNMGR_STATUS_CONNECTED)
	{
		bConnPresence = FALSE;
		bFTPPresence = FALSE;
	}
	if( (nIDEvent == CHECKMESSAGESTIMERID) && bConnPresence &&
bFTPPresence)
	{
		m_Edit_Status.SetWindowText(L"Busy!");
		PrepareMessages();
		FTPGetFile();
		GetMessages();
		m_Edit_Status.SetWindowText(L"Done!");
	}
	else if (nIDEvent == CONNREFRESHTIMERID)
	{
		if ( bAutoOn && !bFileTransferProgress) AutoConnection();
	}

	CDialog::OnTimer(nIDEvent);
}


void CUiDlg::AutoConnection()
{
//	int8* acTemp,*acTempBD[BD_ADDR_SIZE];

	int i, j ;
	int cTempNum;
	char ch[8];
	CString ch1;
	int Rssi[15];
	CString cRssi;
	int MaxRssi = 0;
	DWORD *Conn_Status = new unsigned long;
	HANDLE tempHandle;

	TCHAR szBufW[14];

	m_Edit_Status.SetWindowText(_T("Busy!"));

	if(bConnPresence)
	{

		KillTimer( CHECKMESSAGESTIMERID );
		PrepareLocation();
		if ( !FTPDeleteFile() )
			MessageBox( _T("Error in Sending File"), _T("FTP"), MB_OK);
		else ( FTPClose() );

		ConnMgrReleaseConnection(*ConnectionHandle, FALSE);
```

```
        Sleep(6000);

        CloseHandle(ConnectionHandle);

        bConnPresence = FALSE;
}

*ConnectionHandle = ConnMgrApiReadyEvent();

OnInqButton();
if (iTotalDevices==0)
        return;

for (i = 0; i<iTotalDevices; i++)
{
        guBdAddr = Devices_BD_Addr[i];

        guWriteCmd=HCI_CONNECT;
        SetEvent(ghWriteEvent);

        WaitForSingleObject(ghEvent,INFINITE);

        guWriteCmd=HCI_READ_RSSI;
        bRssiInProgress = TRUE;
        SetEvent(ghWriteEvent);

        WaitForSingleObject(ghEvent,INFINITE);
        bRssiInProgress = FALSE;


        if (!(puMatchData->uiBuf[3] | 0x00))
        {
                cTempNum = 0;
        }
        else if ((puMatchData->uiBuf[3] & 0x80))
        {
                cTempNum=puMatchData->uiBuf[3];
                cTempNum=~cTempNum;
                cTempNum&=0x00ff;
                cTempNum+=1;
                cTempNum = -cTempNum;
        }
        else
        {
                cTempNum = puMatchData->uiBuf[3];
        }
```

```
			Rssi[i] = cTempNum;

			_itoa( cTempNum, ch, 10);
			ch1=ch;

			m_RSSI_List.DeleteString(i);
			m_RSSI_List.InsertString(i,ch1);
			//MessageBox(ch1, NULL, MB_OK);

			guWriteCmd = HCI_DISCONNECT;
			SetEvent(ghWriteEvent);
			WaitForSingleObject(ghEvent,INFINITE);
		}

		if(iTotalDevices>0)
		{
			for (i=1;i<iTotalDevices;i++)
			{
				if (Rssi[i]>Rssi[MaxRssi]) MaxRssi = i;
			}

			guBdAddr = Devices_BD_Addr[MaxRssi];

			iDeviceIndex = MaxRssi;
			SetRegistryValues();

			ConnMgrEstablishConnectionSync( ConnectionInfo,
ConnectionHandle,CONNTIMEOUT, Conn_Status);
			if(*Conn_Status == CONNMGR_STATUS_CONNECTED)
			{
				bConnPresence = TRUE;
				PrepareLocation();
				m_InqList.SetCurSel(MaxRssi);
				m_RSSI_List.SetCurSel(MaxRssi);
//				iDeviceIndex = MaxRssi;

				if( !FTPOpen() )
					MessageBox( _T("Error in Opening FTP"), _T("FTP"),
MB_OK);
				else if ( !FTPSendFile() )
					MessageBox( _T("Error in Sending File"), _T("FTP"),
MB_OK);

				GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(FALSE);
```

```
                        GetDlgItem(IDC_DISCONN_BUTTON)-
>EnableWindow(TRUE);
                }
                else
                {
                        MessageBox(_T("Error in Connecting"),_T("Error"),MB_OK);
                        bAutoOn = FALSE;
                        KillTimer(CONNREFRESHTIMERID);
                        GetDlgItem(IDC_CONN_BUTTON)->EnableWindow(TRUE);
                        GetDlgItem(IDC_DISCONN_BUTTON)-
>EnableWindow(FALSE);
                        GetDlgItem(IDC_RSSI_BUTTON)->EnableWindow(TRUE);
                        GetDlgItem(IDC_INQ_BUTTON)->EnableWindow(TRUE);
                        GetDlgItem(IDC_AUTO_BUTTON)->EnableWindow(TRUE);
                }
        }
        m_Edit_Status.SetWindowText(_T("Done!"));
}


void CUiDlg::OnBrowseButton()
{
        CFileDialog BrowseWindow(TRUE, NULL, NULL,
OFN_HIDEREADONLY,_T("All Files (*.*)|*.*||"),this);
        if ( BrowseWindow.DoModal() == IDOK)
                m_Edit_FileName.SetWindowText(BrowseWindow.GetPathName());
}

void CUiDlg::OnFilesendButton()
{

        unsigned short *ch = new unsigned short [200];
        CString temp;
        int n,length,length1;

        m_Edit_FileName.GetWindowText(ch, 100);
        LocalFile = ch;
//      MessageBox(LocalFile,NULL,MB_OK);

        length = LocalFile.GetLength();
        n = LocalFile.ReverseFind('\\');

        temp = MainRemoteFolder;
        temp = temp + REMOTEFILESFOLDER;
/*      length1 = temp.GetLength();
        if( temp.GetAt( length1 -1) !='/' )
```

```
                temp.SetAt( length1 -1, '/') ;
*/
        RemoteFile = temp + LocalFile.Right(length - n - 1);
        RemoteFolder = MainRemoteFolder;
        RemoteFolder += REMOTEFILESFOLDER;
//      MessageBox(RemoteFile,NULL,MB_OK);

//      if( !FTPOpen() )
//              MessageBox( _T("Error in Opening FTP"), _T("FTP"), MB_OK);
//      else
                if ( !FTPSendFile() )
                MessageBox( _T("Error in Sending File"), _T("FTP"), MB_OK);
//      else ( FTPClose() );

        delete ch;
}

bool CUiDlg::FTPOpen()
{
        bFileTransferProgress = TRUE;

        FTPHandle = ScottyFtpCreate();
        if(FTPHandle == NULL)
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

//      MessageBox(_T("Create"),NULL,MB_OK);

        if ( ScottyFtpConnect(FTPHandle,Server_IP_Address) <0)
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

//      MessageBox( _T("Connect"),NULL,MB_OK);

        if ( ScottyFtpLogin(FTPHandle, FTP_UserName, FTP_Password) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

//      MessageBox(_T("Login"),NULL,MB_OK);
        bFileTransferProgress = FALSE;
        bFTPPresence = TRUE;

        return TRUE;
}


bool CUiDlg::FTPClose()
{
        ScottyFtpQuit(FTPHandle);
        ScottyFtpDestroy(FTPHandle);

        bFileTransferProgress = FALSE;
        bFTPPresence = FALSE;
        return TRUE;
}

bool CUiDlg::FTPSendFile()
{
        m_Edit_Status.SetWindowText(_T("Busy!"));
        bFileTransferProgress = TRUE;

        if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

        if ( ScottyFtpPutFile(FTPHandle, LocalFile, RemoteFile) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }
        m_Edit_Status.SetWindowText(_T("Done!"));
        bFileTransferProgress = FALSE;

        return TRUE;
}


bool CUiDlg::PrepareMessages()
{
        RemoteFile = MainRemoteFolder;
        RemoteFile += REMOTEMESSAGESFOLDER;
        RemoteFolder = RemoteFile;

        RemoteFile += cLoginName;
```

```
	LocalFile = LOCALMESSAGESFOLDER;
	LocalFile += cLoginName;
//	MessageBox(LocalFile,L"Localfile",MB_OK);
//	MessageBox(RemoteFolder,L"RemFolder",MB_OK);
//	MessageBox(RemoteFile,L"RemFile",MB_OK);
	return TRUE;
}


bool CUiDlg::FTPGetFile()
{
	m_Edit_Status.SetWindowText(_T("Busy!"));
	bFileTransferProgress = TRUE;

	if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}

	if ( ScottyFtpGetFile(FTPHandle, RemoteFile, LocalFile) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}

	m_Edit_Status.SetWindowText(_T("Done!"));
	bFileTransferProgress = FALSE;

	return TRUE;
}


bool CUiDlg::FTPDeleteFile()
{
	bFileTransferProgress = TRUE;
	if ( ScottyFtpChangeDirectory(FTPHandle, RemoteFolder) <0 )
	{
		bFileTransferProgress = FALSE;
		return FALSE;
	}

	if ( ScottyFtpDeleteFile(FTPHandle, RemoteFile) <0 )
	{
		bFileTransferProgress = FALSE;
```

```
                return FALSE;
        }
        bFileTransferProgress = FALSE;
        return TRUE;
}



void CUiDlg::OnSelchangeRssiList()
{
        int nIndex;

        nIndex = m_RSSI_List.GetCurSel();
        m_InqList.SetCurSel(nIndex);
}

bool CUiDlg::DestinationNetwork()
{
        int i;
        HRESULT DestResult = 1;
//      CString dest = DestinationConnection;

        for(i=0;(DestinationInfo->szDescription != DestinationConnection) &&
(DestResult != NULL);i++)
        {
                DestResult = ConnMgrEnumDestinations(i, DestinationInfo);
        }

        if (DestinationInfo->szDescription != DestinationConnection)
                return FALSE;

//      MessageBox(DestinationInfo->szDescription, NULL, MB_OK);

        ConnectionInfo->dwParams = 1;
        ConnectionInfo->dwFlags = NULL;
        ConnectionInfo->dwPriority = CONNMGR_PRIORITY_USERINTERACTIVE;
        ConnectionInfo->bExclusive = FALSE;
        ConnectionInfo->bDisabled = FALSE;
        ConnectionInfo->guidDestNet = DestinationInfo->guid;

/*      FILE *fg;

        fg = fopen("\\My Documents\\BTFolder\\test.txt","a");
        fprintf(fg, "%ld:%d:%d:",DestinationInfo->guid.Data1,DestinationInfo-
>guid.Data2,
                DestinationInfo->guid.Data3);
        for (i=0;i<8;i++)
                fprintf(fg, "%d:",DestinationInfo->guid.Data4[i]);
        fprintf(fg,"\n");

        fclose(fg);

*/
//      ConnMgrEstablishConnection( ConnectionInfo, ConnectionHandle);
        return TRUE;
}


bool CUiDlg::GetRemoteNames()
{
        char *TempBTAddr = new char[14];
        char *TempRemoteName = new char[20];
        FILE *fp;
        int i;
        unsigned short *tempch = new unsigned short [20];
        CString temp;

        temp = MainRemoteFolder;
        temp += REMOTEMAPPINGFOLDER;
   if ( ScottyFtpChangeDirectory(FTPHandle, temp) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

        LocalFile = LOCALMAPPINGFILE;
        RemoteFile = MainRemoteFolder;
        RemoteFile += REMOTEMAPPINGFILE;
        if ( ScottyFtpGetFile(FTPHandle, RemoteFile, LocalFile) <0 )
        {
                bFileTransferProgress = FALSE;
                return FALSE;
        }

//      bReceivedMapping = TRUE;
        return TRUE;

}
```

```
bool CUiDlg::AssignRemoteNames()
{
        char *TempBTAddr = new char[14];
        char *TempRemoteName = new char[20];
        CString temp;
        FILE *fp;
        int i;
        unsigned short *tempch = new unsigned short [20];

        fp = fopen(LOCALMAPPINGFILE,"r");
        if( fp == NULL )
        {
                if( bProgramBegin )
                {
                        MessageBox(_T("Mapping File not Found"),NULL,MB_OK);
                        bProgramBegin = FALSE;
                }
                return FALSE;
        }

        while(!feof(fp))
        {
                fscanf(fp,"%s%s", TempBTAddr, TempRemoteName);
/*              temp = TempBTAddr;
                MessageBox(temp,_T("Address"),MB_OK);
                temp = TempRemoteName;
                MessageBox(temp,_T("Name"),MB_OK);
*/
                for ( i=0; i<iTotalDevices; i++ )
                {
                        if(strcmp(cDevices_BD_Addr[i],TempBTAddr)==0)
                        {
                                //temp = TempRemoteName;
                                //MessageBox(temp, NULL, MB_OK);
                                //m_InqList.GetText(i, tempch);
                                strcpy(cRemoteNames[i], TempRemoteName);
                                temp = cDevices_BD_Addr[i];
                                temp = temp + " - ";
                                temp = temp + TempRemoteName;
                                m_InqList.DeleteString(i);
                                m_InqList.InsertString(i, temp);
                        }
                }
        }
        fclose(fp);
        delete TempBTAddr;
        delete TempRemoteName;
        return TRUE;
}


void CUiDlg::PrepareLocation()
{
                FILE *fp;

                char *ch = new char [200];
                char *cHostName = new char[80];
                int length;
                struct hostent *HostDetails;
                CString temp;
                char *Host_IP_Address = new char [20];
                struct in_addr address;

//              cLoginName = "Moses";

//              bFileTransferProgress = TRUE;

                LocalFile = LOCALLOCATIONFOLDER;
                strcpy(ch, LOCALLOCATIONFOLDER);

                length = LocalFile.GetLength();
                if( LocalFile.GetAt( length - 1 ) != '\\' )
                {
                        LocalFile.SetAt( length - 1, '\\' ) ;
                        strcat(ch,"\\");
                }

                LocalFile = LocalFile + cLoginName;
                strcat(ch, cLoginName);
//              LocalFile = LocalFile + ".txt";
//              strcat(ch, ".txt");

                gethostname( cHostName, 80);
                HostDetails = gethostbyname(cHostName);

/*              for (int i = 0; HostDetails->h_addr_list[i] != 0; i++)
                {
                        //temp = HostDetails->h_addr_list[i];
                        memcpy(&address, HostDetails->h_addr_list[i], sizeof(struct
in_addr));
```

```
                        ch1 = inet_ntoa(address);
                        temp = ch1;
                        MessageBox(temp, NULL, MB_OK);
                }
*/
                memcpy(&address, HostDetails->h_addr_list[0], sizeof(struct in_addr));
                Host_IP_Address = inet_ntoa(address);
//              temp = Host_IP_Address;
//              MessageBox(temp, NULL, MB_OK);

                fp = fopen(ch,"w");

                fprintf(fp, "%s\t%s\t%s\n", cDevices_BD_Addr[iDeviceIndex],
                        cRemoteNames[iDeviceIndex],
                        Host_IP_Address);

                fclose(fp);

                RemoteFile = MainRemoteFolder;
                RemoteFile += REMOTELOCATIONFOLDER;
                RemoteFolder = RemoteFile;

                RemoteFile = RemoteFile + cLoginName;
//              RemoteFile = RemoteFile + ".txt";

                delete ch;
//              delete cHostName;
}


bool CUiDlg::GetMessages()
{
        FILE *Message_File;
        int LocalNumMessages = 0;
        char *ch = new char[100];
        CString c;

        strcpy(ch, LOCALMESSAGESFOLDER);
        strcat(ch,cLoginName);

        c = ch;
//      MessageBox(c,NULL,MB_OK);

        Message_File = fopen(ch, "r+");

        while( !feof( Message_File ) )
```

```
	{
		if ( fgets( Messages[NumMessages+LocalNumMessages], 100,
Message_File) == NULL )
		{
			fclose( Message_File );
			NumMessages += LocalNumMessages;
			if (LocalNumMessages==0) return FALSE;
			return TRUE;
		}
		LocalNumMessages++;
	}

	NumMessages += LocalNumMessages;
	fclose (Message_File);
	c = ch;
	DeleteFile(c);

	_itoa(NumMessages, ch, 10);

	c = ch;
//	c += "-";
	c += "New";

	if(LocalNumMessages>0)
	{
		m_Button_NewMsgs.SetWindowText(c);
		m_Button_NewMsgs.EnableWindow(TRUE);
	//	MessageBox(L"button set",NULL,MB_OK);
	}

	MessageBeep(0xFFFFFFFF);
	return LocalNumMessages;
}


void CUiDlg::OnNewmsgsButton()
{
	int i;
	CString c;

	for(i=0; i<NumMessages; i++)
	{
		c = Messages[i];
		m_List_NewMsgs.InsertString( MessagesDisplayed + i, c );
	}
```

```
        MessagesDisplayed +=NumMessages;
        NumMessages = 0;
        m_Button_NewMsgs.SetWindowText(L"New");
        m_Button_NewMsgs.EnableWindow(FALSE);
}


bool CUiDlg::CreateCOMPort()
{
        if(COM_File != NULL)
                CloseHandle(COM_File);
        COM_File = CreateFile(L"COM8:", GENERIC_READ | GENERIC_WRITE,
                0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

//      SetCommParameters (CBR_19200, 8, NOPARITY, ONESTOPBIT);
/*      SetupComm (COM_File, 16, 16);
        COMMTIMEOUTS commTimeOuts;
//      commTimeOuts.ReadIntervalTimeout = iReadTimeout;
        commTimeOuts.ReadTotalTimeoutMultiplier = 0;
        commTimeOuts.ReadTotalTimeoutConstant = 0;
        commTimeOuts.WriteTotalTimeoutMultiplier = 0;
        commTimeOuts.WriteTotalTimeoutConstant = 0;
        SetCommTimeouts(COM_File, &commTimeOuts);
        EscapeCommFunction (COM_File, CLRIR);
        DCB dcb ;
        dcb.DCBlength = sizeof(DCB) ;
        GetCommState(COM_File, &dcb ) ;
        dcb.BaudRate = 19200;
        dcb.ByteSize = 8;
        dcb.Parity = 0;
        dcb.StopBits = 0;
        dcb.fRtsControl = RTS_CONTROL_ENABLE;
        dcb.fDtrControl = DTR_CONTROL_ENABLE;
        dcb.fInX = FALSE;
        dcb.fOutX = FALSE;
        dcb.XonLim = 0;
        dcb.XoffLim = 0;
        dcb.fBinary = TRUE;
        dcb.fParity = TRUE;
        SetCommState(COM_File, &dcb);


//      SetCommParameters (CBR_19200, 8, NOPARITY, ONESTOPBIT);
*/
        if(COM_File == NULL)
                return FALSE;
```

```
	return TRUE;
}


bool CUiDlg::SetRegistryValues()
{
	HKEY hKey;
	int i, length;
	uint8 iDlc = 0x01;
//	bc_subscribe_service_rsp_data *puServiceRspData = NULL;
	char *ch = REGISTRYKEY;
	unsigned short *lpSubKey = new unsigned short[50];
	length = strlen(ch);
	CString c;

	for (i=0;i<=length;i++)
		*(lpSubKey+i) = ch[i];
	c = lpSubKey;
//	MessageBox(c,0,MB_OK);
//	memcpy(lpSubKey, "Drivers\\BuiltIn\\BTSerialCe2", 50);

	RegOpenKeyEx( HKEY_LOCAL_MACHINE, lpSubKey, 0, 0, &hKey );

	RegSetValueEx( hKey, L"BDAddr", NULL, REG_BINARY,
(uint8*)&Devices_BD_Addr[iDeviceIndex], sizeof(bd_addr) );

	RegSetValueEx( hKey, L"Dlc", NULL, REG_BINARY, (uint8*)&iDlc,
sizeof(uint8) );

	RegCloseKey(hKey);
	RegCloseKey(HKEY_LOCAL_MACHINE);

	delete lpSubKey;
	delete ch;

	return TRUE;
}
```

Header Files:

CLoginDlg.h:

```
#if
!defined(AFX_CLOGINDLG_H__E219AB2A_6844_4C5D_94BC_87196396FD06__IN
CLUDED_)
#define
AFX_CLOGINDLG_H__E219AB2A_6844_4C5D_94BC_87196396FD06__INCLUDE
D_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// CLoginDlg.h : header file
//

//#include "Globals.h"

/////////////////////////////////////////////////////////////////////////////
// CLoginDlg dialog

class CLoginDlg : public CDialog
{
// Construction
public:
	CLoginDlg(CWnd* pParent = NULL);   // standard constructor
	bool CheckLogin();
	bool CheckLoginData();


// Dialog Data
	//{{AFX_DATA(CLoginDlg)
	enum { IDD = IDD_LOGIN_DIALOG };
	CEdit	m_Edit_UserName;
	CEdit	m_Edit_Password;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CLoginDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL
// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CLoginDlg)
	afx_msg void OnLogin();
	virtual void OnCancel();
	virtual void OnOK();
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_CLOGINDLG_H__E219AB2A_6844_4C5D_94BC_87196396FD06__IN
CLUDED_)
```

RssiTest.h:

```
#ifndef RSSITEST_H
#define RSSITEST_H


//standard includes
#include <stdio.h>
#include <windows.h>
#include <memory.h>

// other includes
#include <BTDefines.h>
#include <UTILS_Protos.h>
#include <BTApi.h>
//#include "Globals.h"

// HCI specific Initialisation

#define MAX_EVENTS            20
#define HCI_DEFAULT           1
#define HCI_USER_TERMINATE     0x13
#define INQUIRY_DURATION      4 // 10 * 1.28s
#define GIAC             0x9e8b33
#define HCI_EVENTS_REGISTERED  0x6
#define RSSI_TIMEOUT                 2560
#define RSSI_INVALID_VALUE           -128

#define MAXNUMDEVICES 5


typedef enum
{
        EXIT                       = 0,
        HCI_INQUIRY          = 1,
        HCI_CONNECT       ,
        HCI_DISCONNECT,
        HCI_READ_RSSI,
        HCI_READ_REMOTE_NAME
}app_hci_cmds;



typedef hci_handle HCI_Handle;
typedef struct
{
        uint8 uiBuf[60];
        uint8 uiNumber;
} MData;


/**************Global Variables***********************************/
EXTERN HCI_Handle ghDefaultHandle;
EXTERN HCI_Handle ghConnHandle;
EXTERN HCI_Handle ghScoHandle;
EXTERN hci_link_type guScoLinkType;
EXTERN bt_inq_res gauInquiryResult[HCI_MAX_NUM_INQ_RES];
EXTERN int giNumDevice ;
EXTERN bt_device_context ghDeviceContext;
EXTERN bd_addr guBdAddr;

EXTERN char *RemoteName;
EXTERN int iDeviceIndex;
EXTERN bd_addr Devices_BD_Addr[MAXNUMDEVICES];
EXTERN int iTotalDevices;


EXTERN uint16 giStatus;
EXTERN HANDLE ghEvent;
EXTERN HANDLE ghWriteEvent;
EXTERN HANDLE ghExitEvent;
EXTERN HANDLE ghStopEvent ;
EXTERN app_hci_cmds guWriteCmd;
EXTERN HANDLE ghWriteThread;
EXTERN HANDLE ghRssiThread;
EXTERN int giConnFlag;
EXTERN int giProgressFlag;
EXTERN MData* puMatchData;
EXTERN int iAppFlag;
EXTERN uint8 bRssiInProgress ;
EXTERN BOOL ghStopFlag;



/************** Internal Fuction Prototypes***********************/
EXTERN int HciTestIntitialise(void );
EXTERN void APP_HciTestCommands(int iCmdType,void* pvMatchData);
```

```
void APP_InquiryCallback(HCI_Handle hHciHandle,uint32 uEventId, uint8
*pcData,uint32 iLength, void *pvMatchData);
void APP_ConnectionCallback(HCI_Handle hConn,uint32 uEventCode,void
*pcData,uint32 iLength,void *pvMatchData);
void APP_RemoteNameCallback(HCI_Handle hConn,uint32 uEventCode,void
*pcData,uint32 iLength,void *pvMatchData);
void Util_LogBdAddr(FILE *fPtr,bd_addr uBdAddr);
EXTERN DWORD WriterThread(LPVOID pVoid);
EXTERN DWORD RssiThread(LPVOID pVoid);


#endif //RSSITEST_H
```

ui.h:

```
// ui.h : main header file for the UI application
//

#if
!defined(AFX_UI_H__CC175948_4700_4C97_8D84_92A8342AA1B3__INCLUDED_)
#define AFX_UI_H__CC175948_4700_4C97_8D84_92A8342AA1B3__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif// _MSC_VER >= 1000

#ifndef __AFXWIN_H
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols
#include <windowsx.h>
#include "Globals.h"



//////////////////////////////////////////////////////////////////////
// CUiApp:
// See ui.cpp for the implementation of this class
//


class CUiApp : public CWinApp
{
public:
        CUiApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CUiApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation

        //{{AFX_MSG(CUiApp)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
```

```
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft eMbedded Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_UI_H__CC175948_4700_4C97_8D84_92A8342AA1B3__INCLUDED_)
```

uiDlg.h:

```
// uiDlg.h : header file
//
//#include "SDAP_InternalTypes.h"
//#include "SdapStack.h"
//#include "F:\\Moses\\Documents\\DOE\\Programs\\PDA\\Includes\\Sdap_Precom.h"
#include "FTPAPI.h"
#include <connmgr.h>
#include "RssiTest.h"
#include "Globals.h"
#include "ServDisc.h"


#if
!defined(AFX_UIDLG_H__6E8DF798_0BA8_44A5_A421_F2CC9D24432C__INCLU
DED_)
#define
AFX_UIDLG_H__6E8DF798_0BA8_44A5_A421_F2CC9D24432C__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#define CONNREFRESHTIMERID 1
#define CONNREFRESHTIMEOUT 60000

#define CHECKMESSAGESTIMERID 2
#define CHECKMESSAGESTIMEOUT 30000

#define FTPIPADDRESS "139.78.9.151"
#define FTPUSERNAME "vlab"
#define FTPPASSWORD "mstm"

#define CONFIGFILE "\\My Documents\\BTFOLDER\\Configuration\\VLab_Conf.txt"
#define LOCALFILESFOLDER "\\My Documents\\BTFolder\\Files\\"
#define REMOTEFILESFOLDER "/Files/"

#define REMOTEINSTRUCTORSFOLDER "/Instructors/"
#define REMOTELOCATIONFOLDER "/Location/"
#define LOCALLOCATIONFOLDER "\\My Documents\\BTFolder\\Location\\"

#define REMOTEMAPPINGFOLDER "/Mapping/"
#define REMOTEMAPPINGFILE "/Mapping/Mapping.txt"
#define LOCALMAPPINGFILE "\\My Documents\\BTFolder\\Mappings\\Mapping.txt"

#define REMOTELOGINSFOLDER "/Logins/"
```

```
#define REMOTELOGINSFILE "/Logins/Logins.txt"
#define LOCALLOGINSFILE "\\My Documents\\BTFolder\\Logins\\Logins.txt"

#define REMOTEMESSAGESFOLDER "/Messages/"
#define LOCALMESSAGESFOLDER "\\My Documents\\BTFOLDER\\Messages\\"

#define DESTCONNMANAGER "vlab"
#define CONNTIMEOUT 30000

#define LOGINLENGTH 30
#define PASSWORDLENGTH 30

#define ENCRYPTKEY 173

#define MAXNUMMESSAGES 50

#define REGISTRYKEY "Drivers\\BuiltIn\\BTSerialCe2"



EXTERN char *cLoginName;
EXTERN char *cPassword;

/////////////////////////////////////////////////////////////////////////////
// CUiDlg dialog

class CUiDlg : public CDialog
{
// Construction
public:
	CUiDlg(CWnd* pParent = NULL);	// standard constructor
	void AddToListBox(unsigned char* szBuf);
	void AutoConnection();
	bool FTPOpen();
	bool FTPClose();
	bool FTPSendFile();
	bool FTPDeleteFile();
	bool DestinationNetwork();
	bool GetRemoteNames();
	bool AssignRemoteNames();
	void PrepareLocation();
	bool PrepareMessages();
	bool FTPGetFile();
	bool GetMessages();
	bool CreateCOMPort();
	bool SetRegistryValues();
	bool bConnPresence;
	bool bAutoOn;
	bool bFileSelect;
	bool bFileTransferProgress;
	bool bFTPPresence;
	bool bProgramBegin;
	CString LocalFile;
	CString RemoteFile;
	CString RemoteFolder;
	CString MainRemoteFolder;
	CString DestinationConnection;
	HANDLE *ConnectionHandle;
	CONNMGR_CONNECTIONINFO *ConnectionInfo;
	CONNMGR_DESTINATION_INFO *DestinationInfo;
	char *cDevices_BD_Addr[MAXNUMDEVICES];
	char *cRemoteNames[MAXNUMDEVICES];
	HFTP FTPHandle;
	CString FTP_UserName;
	CString FTP_Password;
	CString Server_IP_Address;
	int NumMessages;
	char *Messages[MAXNUMMESSAGES];
	int MessagesDisplayed;
	HANDLE COM_File;



//	bt_device_context BT_Stack_Handle;
//	SDAP_Handle BT_Sdap_Handle;


// Dialog Data
	//{{AFX_DATA(CUiDlg)
	enum { IDD = IDD_UI_DIALOG };
	CListBox	m_RSSI_List;
	CListBox	m_InqList;
	CListBox	m_Msgs_List;
	CListBox	m_List_NewMsgs;
	CButton		m_Button_NewMsgs;
	CEdit	m_Edit_FileName;
	CEdit	m_Edit_Status;
	CProgressCtrl m_InqProgress;
	char m_iAvgRssiVal ;
	CString		m_BDAddr;
	CString		m_RSSIval;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUiDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CUiDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnInqButton();
	afx_msg void OnRssiButton();
	afx_msg void OnExitButton();
	afx_msg void OnClose();
	afx_msg void OnConnButton();
	afx_msg void OnDisconnButton();
	afx_msg void OnAutoButton();
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnFilesendButton();
	afx_msg void OnBrowseButton();
	afx_msg void OnSelchangeRssiList();
	afx_msg void OnNewmsgsButton();
	afx_msg void OnSelchangeInqList();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft eMbedded Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_UIDLG_H__6E8DF798_0BA8_44A5_A421_F2CC9D24432C__INCLU
DED_)
```

APPENDIX E

Code for the Configuration Utility

Program Files:

VLab_Config.cpp:

```
// VLab_Config.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "VLab_Config.h"
#include "VLab_ConfigDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigApp

BEGIN_MESSAGE_MAP(CVLab_ConfigApp, CWinApp)
	//{{AFX_MSG_MAP(CVLab_ConfigApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigApp construction

CVLab_ConfigApp::CVLab_ConfigApp()
	: CWinApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CVLab_ConfigApp object
```

```
CVLab_ConfigApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigApp initialization

BOOL CVLab_ConfigApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

	CVLab_ConfigDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

VLab_ConfigDlg.cpp:

```
// VLab_ConfigDlg.cpp : implementation file
//

#include "stdafx.h"
#include "VLab_Config.h"
#include "VLab_ConfigDlg.h"
#include <stdio.h>
#include <string.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define LOCALFOLDER "\\My Documents"
#define FILENAME "\\VLab_Conf.txt"

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigDlg dialog

CVLab_ConfigDlg::CVLab_ConfigDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CVLab_ConfigDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CVLab_ConfigDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);


}

void CVLab_ConfigDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);

	//{{AFX_DATA_MAP(CVLab_ConfigDlg)
	DDX_Control(pDX, IDC_EDIT5, m_RemFolder);
	DDX_Control(pDX, IDC_EDIT4, m_FTPPassword);
	DDX_Control(pDX, IDC_EDIT3, m_FTPUser);
	DDX_Control(pDX, IDC_EDIT1, m_IPAddress);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CVLab_ConfigDlg, CDialog)
	//{{AFX_MSG_MAP(CVLab_ConfigDlg)
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
	ON_BN_CLICKED(IDC_BUTTON4, OnButton4)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigDlg message handlers

BOOL CVLab_ConfigDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	CenterWindow(GetDesktopWindow());	// center to the hpc screen

	// TODO: Add extra initialization here

	CVLab_ConfigDlg::OnButton4();

	return TRUE;  // return TRUE  unless you set the focus to a control


}



void CVLab_ConfigDlg::OnButton1()
{
	int i;

	unsigned short *IPAddr;

	unsigned short *FTPUser;

	unsigned short *FTPPassword;

	unsigned short *RemFolder;

	IPAddr = new unsigned short[20];
	FTPUser = new unsigned short[20];
	FTPPassword = new unsigned short[20];
	RemFolder = new unsigned short[20];

	// TODO: Add your control notification handler code here
	//CreateFile (TEXT("\\My Documents\\CONFIGFILE.TXT"),	// Open
CONFIGFILE.TXT
  //		GENERIC_READ,		// Open for reading
  //		FILE_SHARE_READ,	// Share for reading
  //		NULL,			// No security
  //		OPEN_EXISTING,		// Existing file only
  //		FILE_ATTRIBUTE_NORMAL,	// Normal file
  //		NULL);					// No template file

	m_IPAddress.GetWindowText(IPAddr, 20);
	m_FTPUser.GetWindowText(FTPUser, 20);
	m_FTPPassword.GetWindowText(FTPPassword, 20);
	m_RemFolder.GetWindowText(RemFolder, 20);



	FILE *stream;

	char *str;
	str = new char[20];

	strcpy(str,LOCALFOLDER);
	strcat(str,FILENAME);

	stream = fopen( str, "w" );
	if (stream == NULL) return;


//	MessageBox(IPAddr,NULL, MB_OK);

	for (i=0; IPAddr[i]!=NULL ;i++)
  fprintf( stream, "%c", IPAddr[i]);

	fprintf( stream, "\n");
	for (i=0; FTPUser[i]!=NULL ;i++)
  fprintf( stream, "%c", FTPUser[i]);

	fprintf( stream, "\n");
	for (i=0; FTPPassword[i]!=NULL ;i++)
  fprintf( stream, "%c", FTPPassword[i]);
```

```
        fprintf( stream, "\n");
        for (i=0; RemFolder[i]!=NULL ;i++)
    fprintf( stream, "%c", RemFolder[i]);

//      fputs(IPAddr, stream);
    fclose( stream );


}

void CVLab_ConfigDlg::OnButton2()
{
        // TODO: Add your control notification handler code here
        CString IPAddr;
        IPAddr ="";
        m_IPAddress.SetWindowText(IPAddr);

        CString FTPUser;
        FTPUser ="";
        m_FTPUser.SetWindowText(FTPUser);

        CString FTPPassword;
        FTPPassword ="";
        m_FTPPassword.SetWindowText(FTPPassword);

        CString RemFolder;
        RemFolder ="";
        m_RemFolder.SetWindowText(RemFolder);


}

void CVLab_ConfigDlg::OnButton4()
{
        // TODO: Add your control notification handler code here

        FILE *stream;
        char *IP;
        IP = new char[20];

        char *User;
        User = new char[20];

        char *Pass;
        Pass = new char[20];
```

```
	char *Rem;
	Rem = new char[20];

	char *str;
	str = new char[20];

	strcpy(str, LOCALFOLDER);
	strcat(str, FILENAME);

	stream = fopen( str, "r" );
	if (stream == NULL)
	{
		fclose(stream);
		return;
	}

	fscanf (stream, "%s", IP);
	fscanf (stream, "%s", User);
	fscanf (stream, "%s", Pass);
	fscanf (stream, "%s", Rem);

	fclose(stream);

	CString IPAddr;
	IPAddr = IP;
	m_IPAddress.SetWindowText(IPAddr);

	CString FTPUser;
	FTPUser = User;
	m_FTPUser.SetWindowText(FTPUser);

	CString FTPPassword;
	FTPPassword = Pass;
	m_FTPPassword.SetWindowText(FTPPassword);

	CString RemFolder;
	RemFolder = Rem;
	m_RemFolder.SetWindowText(RemFolder);

}
```

Header Files:

VLab_Config.h:

```
// VLab_ConfigDlg.cpp : implementation file
//

#include "stdafx.h"
#include "VLab_Config.h"
#include "VLab_ConfigDlg.h"
#include <stdio.h>
#include <string.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define LOCALFOLDER "\\My Documents"
#define FILENAME "\\VLab_Conf.txt"

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigDlg dialog

CVLab_ConfigDlg::CVLab_ConfigDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CVLab_ConfigDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CVLab_ConfigDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);


}

void CVLab_ConfigDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);

	//{{AFX_DATA_MAP(CVLab_ConfigDlg)
	DDX_Control(pDX, IDC_EDIT5, m_RemFolder);
	DDX_Control(pDX, IDC_EDIT4, m_FTPPassword);
	DDX_Control(pDX, IDC_EDIT3, m_FTPUser);
	DDX_Control(pDX, IDC_EDIT1, m_IPAddress);
	//}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CVLab_ConfigDlg, CDialog)
	//{{AFX_MSG_MAP(CVLab_ConfigDlg)
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
	ON_BN_CLICKED(IDC_BUTTON4, OnButton4)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigDlg message handlers

BOOL CVLab_ConfigDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	CenterWindow(GetDesktopWindow());	// center to the hpc screen

	// TODO: Add extra initialization here

	CVLab_ConfigDlg::OnButton4();

	return TRUE;  // return TRUE  unless you set the focus to a control


}



void CVLab_ConfigDlg::OnButton1()
{
	int i;

	unsigned short *IPAddr;

	unsigned short *FTPUser;

	unsigned short *FTPPassword;

	unsigned short *RemFolder;
	IPAddr = new unsigned short[20];
	FTPUser = new unsigned short[20];
	FTPPassword = new unsigned short[20];
	RemFolder = new unsigned short[20];

	// TODO: Add your control notification handler code here
	//CreateFile (TEXT("\\My Documents\\CONFIGFILE.TXT"),      // Open
CONFIGFILE.TXT
  //		GENERIC_READ,         // Open for reading
  //		FILE_SHARE_READ,      // Share for reading
  //		NULL,                 // No security
  //		OPEN_EXISTING,        // Existing file only
  //		FILE_ATTRIBUTE_NORMAL,  // Normal file
  //		NULL);                                  // No template file

	m_IPAddress.GetWindowText(IPAddr, 20);
	m_FTPUser.GetWindowText(FTPUser, 20);
	m_FTPPassword.GetWindowText(FTPPassword, 20);
	m_RemFolder.GetWindowText(RemFolder, 20);



	FILE *stream;

	char *str;
	str = new char[20];

	strcpy(str,LOCALFOLDER);
	strcat(str,FILENAME);

	stream = fopen( str, "w" );
	if (stream == NULL) return;


//	MessageBox(IPAddr,NULL, MB_OK);

	for (i=0; IPAddr[i]!=NULL ;i++)
  fprintf( stream, "%c", IPAddr[i]);

	fprintf( stream, "\n");
	for (i=0; FTPUser[i]!=NULL ;i++)
  fprintf( stream, "%c", FTPUser[i]);

	fprintf( stream, "\n");
	for (i=0; FTPPassword[i]!=NULL ;i++)
```

```
   fprintf( stream, "%c", FTPPassword[i]);

         fprintf( stream, "\n");
         for (i=0; RemFolder[i]!=NULL ;i++)
   fprintf( stream, "%c", RemFolder[i]);

//       fputs(IPAddr, stream);
   fclose( stream );


}

void CVLab_ConfigDlg::OnButton2()
{
        // TODO: Add your control notification handler code here
        CString IPAddr;
        IPAddr ="";
        m_IPAddress.SetWindowText(IPAddr);

        CString FTPUser;
        FTPUser ="";
        m_FTPUser.SetWindowText(FTPUser);

        CString FTPPassword;
        FTPPassword ="";
        m_FTPPassword.SetWindowText(FTPPassword);

        CString RemFolder;
        RemFolder ="";
        m_RemFolder.SetWindowText(RemFolder);


}

void CVLab_ConfigDlg::OnButton4()
{
        // TODO: Add your control notification handler code here

        FILE *stream;
        char *IP;
        IP = new char[20];

        char *User;
        User = new char[20];

        char *Pass;
```

```
Pass = new char[20];

char *Rem;
Rem = new char[20];

char *str;
str = new char[20];

strcpy(str, LOCALFOLDER);
strcat(str, FILENAME);

stream = fopen( str, "r" );
if (stream == NULL)
{
        fclose(stream);
        return;
}

fscanf (stream, "%s", IP);
fscanf (stream, "%s", User);
fscanf (stream, "%s", Pass);
fscanf (stream, "%s", Rem);

fclose(stream);

CString IPAddr;
IPAddr = IP;
m_IPAddress.SetWindowText(IPAddr);

CString FTPUser;
FTPUser = User;
m_FTPUser.SetWindowText(FTPUser);

CString FTPPassword;
FTPPassword = Pass;
m_FTPPassword.SetWindowText(FTPPassword);

CString RemFolder;
RemFolder = Rem;
m_RemFolder.SetWindowText(RemFolder);
```

VLab_ConfigDlg.h:

```
// VLab_ConfigDlg.h : header file
//

#if
!defined(AFX_VLAB_CONFIGDLG_H__2B083A29_8233_4982_B2C2_E0882CEED
AD4__INCLUDED_)
#define
AFX_VLAB_CONFIGDLG_H__2B083A29_8233_4982_B2C2_E0882CEEDAD4__IN
CLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CVLab_ConfigDlg dialog

class CVLab_ConfigDlg : public CDialog
{
// Construction
public:
        CVLab_ConfigDlg(CWnd* pParent = NULL);      // standard constructor


// Dialog Data
        //{{AFX_DATA(CVLab_ConfigDlg)
        enum { IDD = IDD_VLAB_CONFIG_DIALOG };
        CEdit   m_RemFolder;
        CEdit   m_FTPPassword;
        CEdit   m_FTPUser;
        CEdit   m_IPAddress;
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CVLab_ConfigDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CVLab_ConfigDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnButton1();
        afx_msg void OnKillfocusEdit1();
        afx_msg void OnChangeEdit1();
        afx_msg void OnButton2();
        afx_msg void OnButton4();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft eMbedded Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_VLAB_CONFIGDLG_H__2B083A29_8233_4982_B2C2_E0882CEED
AD4 _INCLUDED_)
```

APPENDIX F

Code for the Instructor Access Management Utility

Program Files:

LoginDialog.cpp

```
// LoginDialog.cpp : implementation file
//

#include "stdafx.h"
#include "user_manage.h"
#include "LoginDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define ADMINUSERNAME "admin"
#define ADMINPWD "virtualbluetooth"

/////////////////////////////////////////////////////////////////////////////
// CLoginDialog dialog


CLoginDialog::CLoginDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CLoginDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CLoginDialog)
	//}}AFX_DATA_INIT
}


void CLoginDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CLoginDialog)
	DDX_Control(pDX, IDC_Login_Pwd, m_Login_Pwd);
	DDX_Control(pDX, IDC_Login_Name, m_Login_Name);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CLoginDialog, CDialog)
        //{{AFX_MSG_MAP(CLoginDialog)
        ON_BN_CLICKED(IDC_LOGIN, OnLogin)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CLoginDialog::OnInitDialog()
{
        CDialog::OnInitDialog();
        m_Login_Pwd.SetPasswordChar('*');
        return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CLoginDialog message handlers

void CLoginDialog::OnLogin()
{
        // TODO: Add your control notification handler code here
        LPTSTR ch1 = new char[20];
        LPTSTR ch2 = new char[20];
        m_Login_Name.GetWindowText(ch1,20);
        m_Login_Pwd.GetWindowText(ch2,20);
        CString a = ADMINUSERNAME;
        CString b = ADMINPWD;
        if ((ch1 == a) && (ch2 == b))
        CDialog:OnOK();
        else
        {
                MessageBox(_T("Login name or password is not correct.Please try
again!"),_T("Error Message"),MB_OK);
                m_Login_Name.SetWindowText("");
                m_Login_Pwd.SetWindowText("");
        }
}

void CLoginDialog::OnCancel()
{
        // TODO: Add extra cleanup here

        CDialog::OnCancel();
}
```

user_manage.cpp:

```
// user_manage.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "user_manage.h"
#include "user_manageDlg.h"
#include "LoginDialog.h"
#include <afxwin.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CUser_manageApp

BEGIN_MESSAGE_MAP(CUser_manageApp, CWinApp)
	//{{AFX_MSG_MAP(CUser_manageApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUser_manageApp construction

CUser_manageApp::CUser_manageApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CUser_manageApp object

CUser_manageApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CUser_manageApp initialization

BOOL CUser_manageApp::InitInstance()
```

```
{
        if (!AfxSocketInit())
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }

        AfxEnableControlContainer();


        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared
DLL
#else
        Enable3dControlsStatic();       // Call this when linking to MFC statically
#endif

        int nResponse;

        CLoginDialog Logdlg;

        nResponse = Logdlg.DoModal();
        if (nResponse   = IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
//              MessageBox(NULL,_T("success"),0,MB  OK);
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
                exit(0);
        }

        CUser_manageDlg dlg;
        m  pMainWnd = &dlg;
        nResponse = dlg.DoModal();

/*      if (nResponse -   IDOK)
```

```
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
}
else if (nResponse == IDCANCEL)
{
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
}

// Since the dialog has been closed, return FALSE so that we exit the
//  application, rather than start the application's message pump.
//return FALSE;*/
return FALSE;
```

user_manageDlg.cpp:

```
// user_manageDlg.cpp : implementation file
//

#include "stdafx.h"
#include "user_manage.h"
#include "user_manageDlg.h"
#include <afxinet.h>
#include <afxwin.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
```

```
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUser_manageDlg dialog

CUser_manageDlg::CUser_manageDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CUser_manageDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CUser_manageDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	int i;
	for (i=0;i<30;i++)
	{
		Name[i] = new char [30];
		Password[i] = new char [30];
		Fullname[i] = new char [50];
	}
	n_users = 0;
	selected_user = -1;

}

void CUser_manageDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CUser_manageDlg)
	DDX_Control(pDX, IDC_LIST, m_List_LoginList);
	DDX_Control(pDX, IDC_ADD, m_Button_Add);
	DDX_Control(pDX, IDC_EDIT_FULLNAME, m_Edit_FullName);
	DDX_Control(pDX, IDC_EDIT_PASSWORD, m_Edit_PassWord);
	DDX_Control(pDX, IDC_EDIT_USERNAME, m_Edit_UserName);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CUser_manageDlg, CDialog)
	//{{AFX_MSG_MAP(CUser_manageDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_ADD, OnAdd)
	ON_BN_CLICKED(IDC_DELETE, OnDelete)
	ON_LBN_SELCHANGE(IDC_LIST, OnList)
	ON_BN_CLICKED(IDC_OK, OnOk)
	ON_BN_CLICKED(IDC_UPDATE, OnUpdate)
	ON_BN_CLICKED(IDC_APPLY, OnApply)
	ON_BN_CLICKED(IDC_EXIT, OnExit)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUser_manageDlg message handlers

BOOL CUser_manageDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog. The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	FtpConnection();
	Display();

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CUser_manageDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CUser_manageDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM)
dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
```

```
			dc.DrawIcon(x, y, m_hIcon);
		}
		else
		{
			CDialog::OnPaint();
		}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CUser_manageDlg::OnQueryDragIcon()
{
		return (HCURSOR) m_hIcon;
}

void CUser_manageDlg::FtpConnection()
{
		int i=0;
		int length1,length2,length3,total;
		FILE *file;
		char *templine = new char[256];
		char *ServerAddr   new char[20];
		LPCSTR User_Name = new char[20];
		LPCSTR UserPwd = new char[20];
		file    fopen("C:\\serverinfo.txt","r");
		fgets(templine,256,file);
		sscanf(templine,"%s\t%s\t%s",ServerAddr,User_Name,UserPwd);
		CInternetSession CFTP;
		CFtpConnection* FTP_MSTM;
//		char *sfile;
		CString s;
		LPDWORD d;
		char *line = new char[100];
		char *show = new char[1];
		d = new DWORD;
		*d=250;
		char *cha = new char[30];
		CString des;

		n  users = 0;

		FTP_MSTM = CFTP.GetFtpConnection( ServerAddr, User_Name, UserPwd, 21,
TRUE);
//		FTP_MSTM->GetCurrentDirectory(s, d);
		s="/VLabPDA/Logins/";
		FTP_MSTM->SetCurrentDirectory(s);
```

```
	CString gh = "/VLabPDA/Logins/Logins.txt";
	des = "C:\\Logins.txt";
	FTP_MSTM-
>GetFile(gh,des,FILE_ATTRIBUTE_NORMAL,FTP_TRANSFER_TYPE_BINARY,
1);
	FTP_MSTM->Close();

	if ((in_file = fopen("C:\\Logins.txt","r")) == NULL)
	{
		MessageBox(_T("No such file exists"),NULL,MB_OK);
	   return;
	}
	//else
	//	MessageBox(_T("File is opened"),NULL,MB_OK);

   while (!feof(in_file))
	{
		if( fgets(line, 256, in_file) == NULL)
		{
			fclose(in_file);
			MessageBox(_T("No user exits"),NULL,MB_OK);
			if (DeleteFile(des) == 0)
				MessageBox(_T("Error"),NULL,MB_OK);
			return;
		}

	   sscanf(line, "%s\t%s\t%s", Name[n_users],
Password[n_users],Fullname[n_users]);
/*		if (*Name[0] == -51)
	{
		fclose(in_file);
		MessageBox(_T("EOF"),NULL,MB_OK);
		if (DeleteFile(des) == 0)
			MessageBox(_T("Error"),NULL,MB_OK);

		return;
	}
*/		length1 = strlen(Name[n_users]);
/*		for (i=0;i<length1;i++)
		{
			//Name[n_users][i] = Name[n_users][i]^encap;
			*(Name[n_users]+i) = *(Name[n_users]+i);
		}
		length2 = strlen(Password[n_users]);
		for (i=0;i<length2;i++)
		{
				//Password[n_users][i] = Password[n_users][i]^encap;
				*(Password[n_users]+i) = *(Password[n_users]+i);
			}
			length3 = strlen(Fullname[n_users]);
			for (i=0;i<length3;i++)
			{
				//Fullname[n_users][i] = Fullname[n_users][i]^encap;
				*(Fullname[n_users]+i) = *(Fullname[n_users]+i);
			}
*/
			length2 = strlen(Password[n_users]);
			length3 = strlen(Fullname[n_users]);
			total = strlen(Name[n_users]) + strlen(Password[n_users]) +
strlen(Fullname[n_users]);
			//*show	line[total+1];
	/*	while (line[total+2] != '\n')
			{
				*(Fullname[n_users]+length3) = line[total+2];
				total++;
				length3++;
			}*/
			n_users++;
	}
	fclose(in_file);

	DeleteFile(des);


	delete line;
	delete d;
	delete cha;


}
void CUser_manageDlg::Display()
{
	// Decrypt all data using EncryptAllData function and show all data as normal
	// words, then encrypt data again.
	int t;
//	MessageBox(_T("encrypt"),NULL,MB_OK);
	LPCTSTR temp_initial = new char [256];
	//if (*Name[0] == -51) return;
	EncryptAllData();
	for (t=0;t< n_users;t++)
	{
		CString temp_initial = Name[t];
```

```
			temp_initial+= "----------";
			temp_initial+= Password[t];
			temp_initial+= "----------";
			temp_initial+= Fullname[t];
			m_List_LoginList.InsertString(t,temp_initial);
		}
		EncryptAllData();
		selected_user =-1;
}
void CUser_manageDlg::OnAdd()
{
		// TODO: Add your control notification handler code here
		int i=0;
		int length=0;
		LPTSTR ch = new char[20];
		n_users++;

		m_Edit_UserName.GetWindowText(ch,20);

		length = strlen(ch);

		for (i=0;i<=length;i++)
		{
			*(Name[n_users-1]+i) = ch[i];
		}
		//Name[n_users-1][i] = '\0';
		m_Edit_PassWord.GetWindowText(ch,50);
		length = strlen(ch);
		for (i=0;i<=length;i++)
		{
			*(Password[n_users-1]+i) = ch[i];
		}
		//Password[n_users-1][i] = '\0';

		m_Edit_FullName.GetWindowText(ch,50);
		length = strlen(ch);
		for (i=0;i<=length;i++)
		{
			*(Fullname[n_users-1]+i) = ch[i];
		}
		//Fullname[n_users-1][i] = '\0';
		CString temp;
		temp = Name[n_users-1];
		temp+="----------";
		temp+=Password[n_users-1];
		temp+="----------";
```

```
        temp+=Fullname[n_users-1];

        m_List_LoginList.InsertString(n_users-1,temp);
        EncryptData(n_users-1);
//      EncryptData(n_users-1);

//      fprintf(in_file,"\n%s %s %s", Name[n_users],
Password[n_users],Fullname[n_users]);
        m_Edit_UserName.SetWindowText("");
        m_Edit_PassWord.SetWindowText("");
        m_Edit_FullName.SetWindowText("");

}

void CUser_manageDlg::OnDelete()
{
        // TODO: Add your control notification handler code here
        int i;
        if(selected_user = = -1)
                MessageBox(_T("Selecting in error"),NULL,MB_OK);
        char *temp_name;
        char *temp_password;
        char *temp_fullname;
        temp_name = Name[selected_user];
        temp_password = Password[selected_user];
        temp_fullname = Fullname[selected_user];

        for (i=selected_user;i<n_users-1;i++)
        {
                Fullname[i] = Fullname[i+1];
                Name[i] = Name[i+1];
                Password[i] = Password[i+1];
        }
        Name[n_users-1] = temp_name;
        Password[n_users-1] = temp_password;
        Fullname[n_users-1] = temp_fullname;
        n_users = n_users - 1;
        //CUser_manageDlg::Display();
        m_List_LoginList.DeleteString(selected_user);
        m_Edit_UserName.SetWindowText("");
        m_Edit_PassWord.SetWindowText("");
        m_Edit_FullName.SetWindowText("");

}
```

```
void CUser_manageDlg::OnList()
{
	// show the selected user on the left list

	selected_user = m_List_LoginList.GetCurSel();
	if( selected_user == prev_selected_user )
	{
		m_List_LoginList.SetCurSel(-1);// selected_user, FALSE);
		m_Edit_UserName.SetWindowText("");
		m_Edit_PassWord.SetWindowText("");
		m_Edit_FullName.SetWindowText("");
		selected_user = -1;
		prev_selected_user = -1;
	}
	else
	{
		EncryptData(selected_user);
		m_Edit_UserName.SetWindowText(Name[selected_user]);
		m_Edit_PassWord.SetWindowText(Password[selected_user]);
		m_Edit_FullName.SetWindowText(Fullname[selected_user]);
		EncryptData(selected_user);
		prev_selected_user = selected_user;
	}

}

void CUser_manageDlg::OnOk()
{
	// TODO: Add your control notification handler code here
	OnApply();
	int i;
	for (i=0;i<30;i++)
	{
		delete Name[i];
		delete Password[i];
		delete Fullname[i];
	}
	exit(0);
}

void CUser_manageDlg::OnUpdate()
{
	// TODO: Add your control notification handler code here
	CString temp;
	char* ch1 = new char[30];
```

```
        char* ch2 = new char[30];
        char* ch3 = new char[50];
        m_Edit_UserName.GetWindowText(ch1,30);
        memcpy(Name[selected_user], ch1,30);
        m_Edit_PassWord.GetWindowText(ch2,30);
        memcpy(Password[selected_user], ch2,30);
        m_Edit_FullName.GetWindowText(ch3,50);
        memcpy(Fullname[selected_user], ch3,50);
        temp = Name[selected_user];
        temp+="----------";
        temp+=Password[selected_user];
        temp+="----------";
        temp+=Fullname[selected_user];
        m_List_LoginList.DeleteString(selected_user);
        m_List_LoginList.InsertString(selected_user,temp);
        EncryptData(selected_user);
        m_Edit_UserName.SetWindowText("");
        m_Edit_PassWord.SetWindowText("");
        m_Edit_FullName.SetWindowText("");
}

void CUser_manageDlg::OnApply()
{
        // TODO: Add your control notification handler code here
        int i;
//      char *a;

        if ((in_file = fopen("C:\\Logins.txt","w")) == NULL)
                MessageBox(_T("File was not opened"),NULL,MB_OK);
        //EncryptAllData();
        for (i=0;i<n_users;i++)
        {
                if (i == (n_users - 1))
                   fprintf(in_file,"%s\t%s\t%s",Name[i],Password[i],Fullname[i]);
                else
                        fprintf(in_file,"%s\t%s\t%s\n",Name[i],Password[i],Fullname[i]);
        }
        fclose(in_file);
        m_Edit_UserName.SetWindowText("");
        m_Edit_PassWord.SetWindowText("");
        m_Edit_FullName.SetWindowText("");

        FILE *file;
        char *templine = new char[256];
        char *ServerAddr = new char[20];
        LPCSTR User_Name = new char[20];
        LPCSTR UserPwd = new char[20];
        file = fopen("C:\\serverinfo.txt","r");
        fgets(templine,256,file);
        sscanf(templine,"%s\t%s\t%s",ServerAddr,User_Name,UserPwd);
        LPDWORD d;
        CString s;//= new char[250];
        CString des;//new char[250];
//      sfile = new char[250];
        d = new DWORD;
        *d=250;
        CInternetSession CFTP;
        CFtpConnection* FTP_MSTM;

        FTP_MSTM = CFTP.GetFtpConnection( ServerAddr, User_Name, UserPwd, 21,
TRUE);
//      FTP_MSTM->GetCurrentDirectory(s, d);
        s="/VLabPDA/Logins";
        FTP_MSTM->SetCurrentDirectory(s);
   des = "/VLabPDA/Logins/Logins.txt";

        CString h = "C:\\Logins.txt";
        if (FTP_MSTM->PutFile(h,des,FTP_TRANSFER_TYPE_BINARY, 1)==0)
        {
                GetLastError();
                MessageBox(_T("Error"),NULL,MB_OK);
        }
        FTP_MSTM->Close();
        DeleteFile(h);
}

void CUser_manageDlg::EncryptAllData()
{
        // Encrypting all data using encrypting key
        int length,i,j;

        for(i=0;i<n_users;i++)
        {
                length = strlen(Name[i]);
                for (j=0;j<length;j++)
                {
                        *(Name[i]+j) = (*(Name[i]+j)) ^ EncryptKey;
                }
                length = strlen(Password[i]);
                for (j=0;j<length;j++)
                {
                        *(Password[i]+j) = (*(Password[i]+j)) ^ EncryptKey;
			}
			length = strlen(Fullname[i]);
			for (j=0;j<length;j++)
			{
				*(Fullname[i]+j) = (*(Fullname[i]+j)) ^ EncryptKey;
			}
		}
}

void CUser_manageDlg::EncryptData(int iIndex)
{
		int length,j;
		length = strlen(Name[iIndex]);
		for (j=0;j<length;j++)
		{
			*(Name[iIndex]+j) = (*(Name[iIndex]+j)) ^ EncryptKey;
		}
		length = strlen(Password[iIndex]);
		for (j=0;j<length;j++)
		{
			*(Password[iIndex]+j) = (*(Password[iIndex]+j)) ^ EncryptKey;
		}
		length = strlen(Fullname[iIndex]);
		for (j=0;j<length;j++)
		{
			*(Fullname[iIndex]+j) = (*(Fullname[iIndex]+j)) ^ EncryptKey;
		}
}

void CUser_manageDlg::OnExit()
{
		// TODO: Add your control notification handler code here
		exit(0);
}
```

Header Files:

LoginDialog.h:

```
#if
!defined(AFX_LOGINDIALOG_H__9789D8F4_91E2_4F56_BF10_8A212C45C7CB__
INCLUDED_)
#define
AFX_LOGINDIALOG_H__9789D8F4_91E2_4F56_BF10_8A212C45C7CB__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LoginDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CLoginDialog dialog

class CLoginDialog : public CDialog
{
// Construction
public:
	CLoginDialog(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CLoginDialog)
	enum { IDD = IDD_LOGIN_DIALOG };
	CEdit	m_Login_Pwd;
	CEdit	m_Login_Name;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CLoginDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CLoginDialog)
	virtual BOOL OnInitDialog();
	afx_msg void OnLogin();
	virtual void OnCancel();
//	afx_msg void OnLogin();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_LOGINDIALOG_H__9789D8F4_91E2_4F56_BF10_8A212C45C7CB__
INCLUDED_)
```

user_manage.h:

```
// user_manage.h : main header file for the USER_MANAGE application
//

#if
!defined(AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4
__INCLUDED_)
#define
AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CUser_manageApp:
// See user_manage.cpp for the implementation of this class
//

class CUser_manageApp : public CWinApp
{
public:
	CUser_manageApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUser_manageApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CUser_manageApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4
__INCLUDED_)
```

user_manageDlg.h:

```
// user_manage.h : main header file for the USER_MANAGE application
//

#if
!defined(AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4
__INCLUDED_)
#define
AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4__INCL
UDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CUser_manageApp:
// See user_manage.cpp for the implementation of this class
//

class CUser_manageApp : public CWinApp
{
public:
	CUser_manageApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUser_manageApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CUser_manageApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_USER_MANAGE_H__ECD6D520_9B91_42C5_9BE2_8DAE88E2F4E4
__INCLUDED_)
```

2

VITA

Lynn Moses George

Candidate for the Degree of Master of Science

Thesis: Development of the Wireless Instructor System and Bluetooth Handover Technologies for Improved Virtual Laboratory Applications

Major Field: Electrical and Computer Engineering

Biographical:

Education: Received Bachelor of Engineering degree in Electrical and Electronics Engineering from Madurai Kamaraj University, Tamilnadu, India in May 2000. Completed the requirements for the Master of Science degree with a major in Electrical and Computer Engineering at Oklahoma State University in August, 2003.

Experience: Graduate Research Assistant in Electrical and Computer Engineering Department, Oklahoma State University, Stillwater, Oklahoma, August 2001 to July 2003.
System Design Engineer in GDA Technologies, Chennai, India, June 2000 to July 2001.

Professional Membership: IEEE – Student Member