Implementation of Electrical and Electronics' Department's Wireless Local Area Network and Central Server

by

Sim Yih Chun

A project dissertation
submitted in partial fulfilment of
the requirements for the
BACHELOR OF ENGINEERING (Hons)
(Electrical & Electronics Engineering)

DECEMBER 2006

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Implementation of Electrical and Electronics' Department's Wireless Local Area Network and Central Server

by

Sim Yih Chun

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,

[illegible]

(En Azman Zakariya)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

December 2006

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

SIM YIN CHUN
840521-13-5501
Matric No: 3459

ABSTRACT

This project is designed to overcome two related problems: the Electrical and Electronics Department is located on two blocks in the new campus, namely Block 22 and Block 23. Currently these blocks are connected through a physically wired local area network connection under the control of the Information Technology and Media Services (ITMS) department. Unfortunately, this has the restriction of requiring a computer to be physically near a port in order to log onto the network, and locations without a port are essentially cut off from the other computers. Another observed problem is the excessive dependence on unreliable paper records to track students as they borrow electronic components from the stores. To overcome these problems, this project involves the **design and implementation of a wireless network** that covers both the blocks. The project itself will have two stages: first, to set up the hardware for a wireless network in the two blocks, and secondly, to design and implement a centralised system for recording inventory when they are borrowed by students as a demonstration of the possibilities offered by such a network.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, En Azman Zakariya, for his guidance, support and valuable feedback throughout this project. Also, I would like to thank the technicians of the Electrical and Electronics Department of University Teknologi Petronas, especially Kak Siti Hawa and Encik Musa for their invaluable help in this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND NOMENCLATURE

IEEE	Institute of Electrical and Electronics Engineers
AP	Access Point
LAN	Local Area Network
EE	Electrical and Electronics Faculty
TCP/IP	Transmission Control Protocol/Internet Protocol
IP	Internet Protocol
PC	Personal computer
UTP	University of Technology Petronas
API	Application Programming Interface
VB	Visual Basic
MAC	Media Access Control

CHAPTER 1

INTRODUCTION

This chapter provides a brief summary of the project. The project background is explained, with a short explanation on the various wireless technologies that will be used in this project. The problem statement elaborates on the reasons behind the project, as one of the pillars of engineering is the ability to solve problems. This is followed by the objectives, which are the goals that should be achieved once the project has been completed successfully.

1.1 Background

There are currently two main standards for commercial wireless application, both with their own unique advantages and disadvantages:

1. **WiFi**, a wireless Local Area Network compatibility standard based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 specification. Commercial set-ups of WiFi networks rely on the use of Access Points (APs) that the clients connect to, and the most common standard of 802.11g (Wireless-G) has typical speeds of around 20Mbps, even though the theoretical maximum is 54Mbps assuming ideal data transfer conditions. It operates up to a theoretical maximum range of 300m. Recent models have improved this to a maximum of 108Mbps [1], but with reduced range, or with even longer ranges at reduced data transfer rates. A complete WiFi set-up is expensive, with an Access Point costing upwards of RM300, and each WiFi network adapter costing around RM150.

2. **Bluetooth**, a wireless Personal Area Network compatibility standard based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.15.1 specification. Bluetooth utilises the same 2.4GHz frequency band as WiFi, with three different classes based on the power/ range of the Bluetooth device. A

Class 1 Bluetooth device has a power output of 100mW, giving it a theoretical maximum range of approximately 100m, although in practice, the range would be much lower than this [2]. Its main advantage is its low power consumption/ low cost with dongles costing as little as RM50 each. However, it is bandwidth limited, allowing a throughput of up to 1 Mbps, or 125 kB/s.

1.2 Problem Statement

The two main problems identified by this project are listed below:

- Currently, there are no set-ups using wireless technology in the new campus, even though wireless LAN technology is becoming one of the fastest growing fields in telecommunications. Even other projects currently in development have yet to set up any wireless equipment on campus.

- When students borrow electronic components from the lab, the current system involves recording the student's name and borrowed components on a paper form, which is then filed. Even then, different labs use different systems, so the system is actually quite inefficient. A proper, centralised record will speed up the component borrowing process, as well as allowing the technicians to keep track of all the components currently on loan.

1.3 Objectives

Based on the problems stated above, this project has a two-fold objective:

- To set up a wireless network infrastructure in Buildings 22 and 23 of the new campus that will also be connected to the main UTP network.

- To design, program and deploy a centralised inventory record system that keeps track of the electronics components in all the EE labs, with the results stored on a central server.

CHAPTER 2

THEORY AND LITERATURE REVIEW

This chapter explores the theories behind this project in greater detail. The following sections will explain the underlying architecture of the technologies used, as well as the operation of socket programming in Windows using the WinSock API. Most of this chapter will touch on existing information that support the design decisions made in this project.

2.1 Advantages of Wireless Networking

Wireless communications is a rapidly developing field, and technology that involves the transfer of information across radio waves has improved by leaps and bounds in the past decade. There are many advantages of wireless networking [3], namely:

- Increased mobility: with wireless communication, it is possible to remain in contact even if the other person moves between different locations. The wireless device is accessible as long as it was within operational range, which means that the people involved are not constrained by the need to remain in the same place.

- Increased flexibility: wireless communications removes the need for complex wiring, which in turn leads to flexibility in the layout of a network. The design of a wireless network is not limited by the wiring of the building, and it is much easier to add or remove elements of the wireless network compared to a fixed network.

- Aesthetics: another advantage of wireless technology is the lack of unsightly cables makes for a more aesthetically pleasing environment.

2.2 Classifications of Wireless Technologies

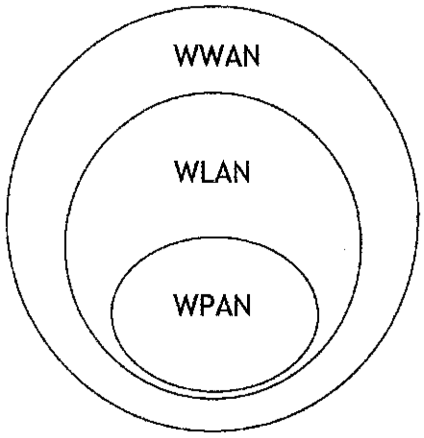


Figure 2.1: Wireless technology categories

Wireless technologies can be divided into three main categories, as shown in Figure 2.1.

1. Wireless Wide Area Networks (WWAN): large coverage, mostly for voice telephony such as cellular companies. Latest generations of the technology include integration of data and video streams.
2. Wireless Local Area Networks (WLAN): medium power and medium coverage. Includes the IEEE 802.11 family of specifications, such as wireless Ethernet (Wi-Fi), Home Radio Frequency, and High-Performance Radio Local Area Networks (HIPERLAN).
3. Wireless Personal Area Networks (WPAN): low-power, short range applications. Includes Bluetooth, IrDA, and other specifications from the IEEE 802.15 family.

For this project, we will focus on technologies involving WLAN and WPAN applications.

2.3 Comparison Study: Bluetooth versus WiFi

Although both Bluetooth and WiFi are wireless protocols operating in the 2.4 GHz frequency range, there are fundamental differences between the two.

WiFi (IEEE 802.11)

Figure 2.2: The Wi-Fi logo

There are several different standards available for the 802.11 specification: each different flavour is signified by a different alphabet appended to the end, e.g. 802.11a, 802.11b, etc.

The most commonly available standard is Wireless G, or more correctly known as the 802.11g specification. It is compatible with devices designed to follow the 802.11b specification. However, the wireless access bridges in the lab currently utilise the earlier **802.11b standard**. The 802.11b standard uses a variation of direct-sequence spread spectrum (DSSS) modulation techniques which is similar to the modulation used by Code Division Multiple Access (CDMA). Devices using the 802.11b standard have a maximum data throughput of up to 11 Mbps, although operational data rates will be lower than that. With a high gain external omni-directional antenna, 802.11b devices have an operational range of several kilometres through open space, and at least a few hundred meters even through obstacles.

- However, WiFi modulation has been shown to be more **vulnerable to interference**, especially from other radio devices operating in the same frequency band. Also, data rates near the maximum data output is difficult to achieve in practice, due to much higher susceptibilities to interference as the data rate increases.

- Cost is also an important limitation, since it is **quite expensive to set up** even a medium scale network. A single Access Point costs more than RM300, and each client would require network adapters costing between RM100 and RM200 [4]. This means that for a small network of 7 computers, the total cost would be RM900 to RM1500, and the cost climbs even higher when more computers are added. With a project budget of only RM250, this is a critically important factor to consider when comparing the different options available.

- Also, since different manufacturers currently use different techniques to boost the range or data rate of the wireless transmissions, **interoperability is not guaranteed** between different brands. For example, some companies use what is called the Super-G specification, which quadruples the range and doubles the data rate, but is not compatible with devices made by other companies, and have much higher interference potential in the 2.4 GHz spectrum. Even newer standards are being introduced, using multiple-input multiple-output (MIMO) technology to boost the capabilities even further. However, these solutions are too prohibitively expensive for the scope of this project.

- Another weakness of WiFi is the **questionable security**. WiFi networks are secured through Wired Equivalent Privacy (WEP), which is basically a numeric encryption method. However, this has been shown to be crackable when enough data is collected from the transmitter, this compromising the security of the wireless network. WiFi cracking tools are available on the internet, allowing anyone with a receiver to tap into the bandwidth of a secured WiFi network.

- The main advantage of WiFi systems is its ability to **operate reliably across larger distances** compared to Bluetooth. While the reliability of Bluetooth devices deteriorate with range, the higher powered WiFi devices have a significantly longer operational range, and are better at penetrating obstacles. This can be further boosted by deploying a directional antenna.

Bluetooth (IEEE 802.15.1)

Figure 2.3: The Bluetooth logo

Originally designed as a low-power industrial specification for short-range radio applications, Bluetooth has emerged as a low cost, secure protocol of communicating between various different devices, from Personal Digital Assistants (PDA) to cellphones to printers to other personal computers.

- Bluetooth uses adaptive frequency-hopping spread spectrum modulation, which makes it **less susceptible to electromagnetic interference** from other devices in the same radio frequency range. The Bluetooth transceiver chip divides the bandwidth into 79 channels and changes channel 1600 times a second, minimizing the chance of interfering with other transmissions in the same frequency band.

- Bluetooth is a **far more cost-effective option** for this project. A single Bluetooth USB dongle costs RM50, which enables a computer to function as either a client or a server on the Bluetooth network when connected. This means that no access points are required in order to set-up a working network, and the network can be rapidly reconfigured as necessary using only software controls. A network covering 7 computers for a block would cost just RM350 to set up, and its low cost means that maintenance and replacement is much easier.

- The design of Bluetooth allows it to talk to other devices besides computers, so this network can be **easily expanded** in the future to communicate with mobile phones and other peripherals. Many phones and PDA devices support Bluetooth due to its low cost and easy usage, so future applications can focus on delivering services from the EE department to other devices as they pass through the EE department.

- Of course, Bluetooth has its limitations as well. Current specifications of Bluetooth allows a central master to accept connections from a **maximum of 7 slave devices**, which means that the network has to consist of several sub-networks joined by other communications medium. The current design allows for a central master to be connected to an Ethernet connection, and the software side then bridges the connection to other sub-networks. In Bluetooth terms, the sub-network is called a 'piconet' while a collection of piconets is known as a 'scatternet'.

- Another limitation is the **lower data rate and shorter range** of the devices. A typical Bluetooth connection has a maximum data transfer rate of 125 kB/s, with practical rates of around 100 kB/s. This is still fast enough for our applications, which is why it is not a major concern in the implementation of this project. However, it is something to bear in mind since the speed is effectively divided by the number of simultaneous connections, so a fully loaded network might be slower than expected. Range is a more serious issue: even though the theoretical maximum range is 100 m, the low-powered Bluetooth devices experience significant connection problems when the room is enclosed in glass walls (such as in the laboratories) and there are obstacles in the way.

Bearing these factors in mind, it was decided that a mixed wireless network would be the most efficient solution for the project. By combining the advantages of WiFi and Bluetooth, the aim of the project is to provide a high performance wireless network that still remains cost effective.

2.4 Windows Socket Programming

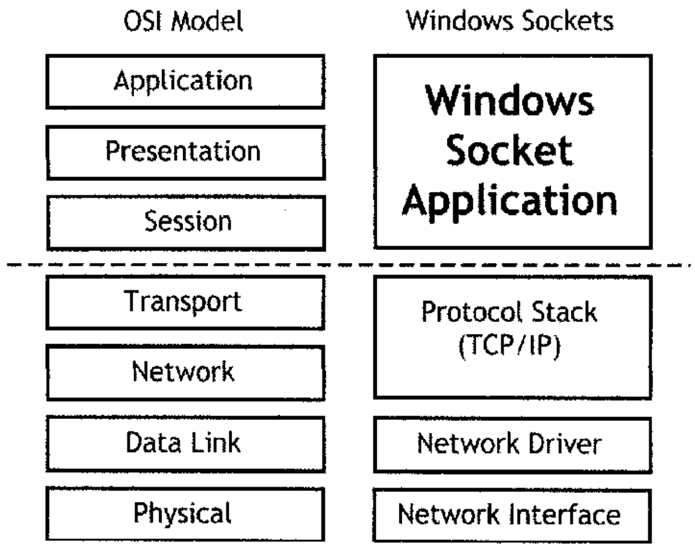


Figure 2.4: The Windows Socket network model

The most common reference model for most networks is the Open Systems Interconnect (OSI) network model, where the entire system is divided into distinct layers that provide services to higher layers. Each layer is connected by allowing predetermined function calls to be used on a lower layer, therefore allowing each layer to be developed independent of the lower layers. This is an important aspect of the OSI network model in this project, as it allows us to develop the software separately from the hardware portion.

Windows sockets applications are actually equivalent to the top three layers in the OSI model [5]. The socket application usually consists of a dynamic link library (DLL) that provides high-level APIs to other applications that perform functions unique to that particular application. A WinSock API (WSA) allows the application to access the network system and transmit information to other points on the network, but the main advantage of WinSock is that it is easy to understand, and is flexible enough to allow many different applications to be developed for it.

CHAPTER 3

PROJECT WORK

This chapter identifies the steps involved in the project work. It follows the logical order of procedures in preparing the project, beginning with an outline of the project layout, followed by step-by-step instructions to set up each hardware component. It also identifies and details the equipment required for the project.

3.1 Project Layout: Communication between blocks

The Electrical and Electronics Engineering Faculty covers two buildings, and this has been demonstrated to be possible when using a pair of WiFi bridges between the two blocks. For shorter ranges, a Wireless G Access Point will be used to expand the coverage. Figure 3.1 shows a sketch of the layout for the connection between blocks.

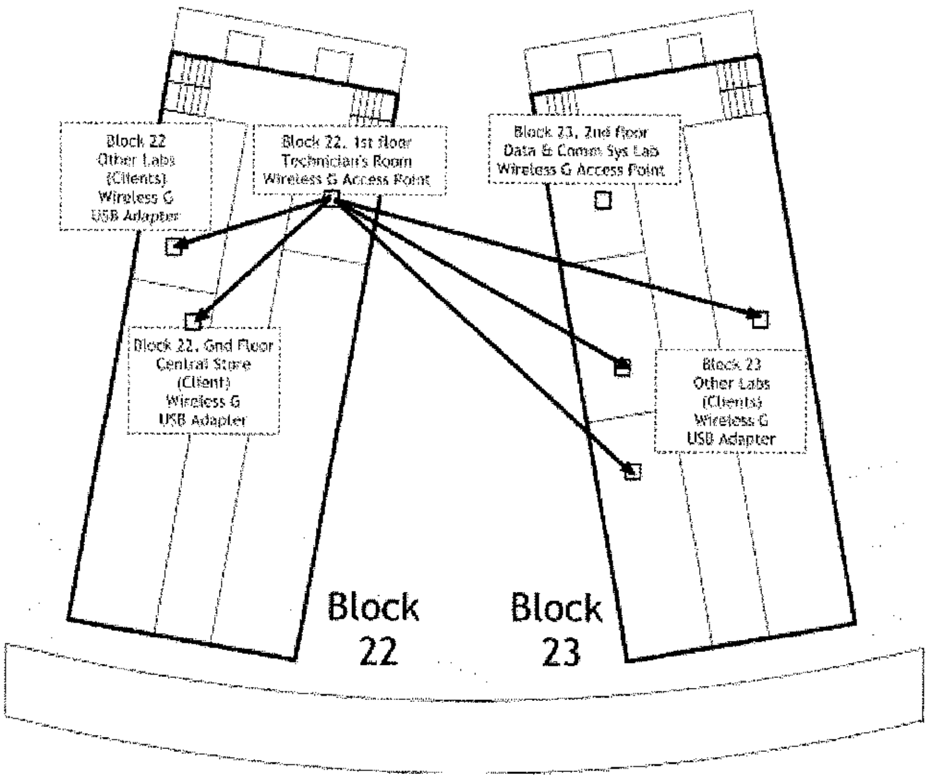


Figure 3.1: Layout of Project Location

3.2 Project Layout: Within a single block

Figure 3.2 shows the hardware layout for the project. The Wireless G access point is the hub for the other client computers, and additional clients may be added using Bluetooth dongles to transmit data. A technician is needed to maintain the main server PC, and ideally, software design ensures that user intervention is minimal in server operation. Other technicians will approve student actions at their respective client PCs.

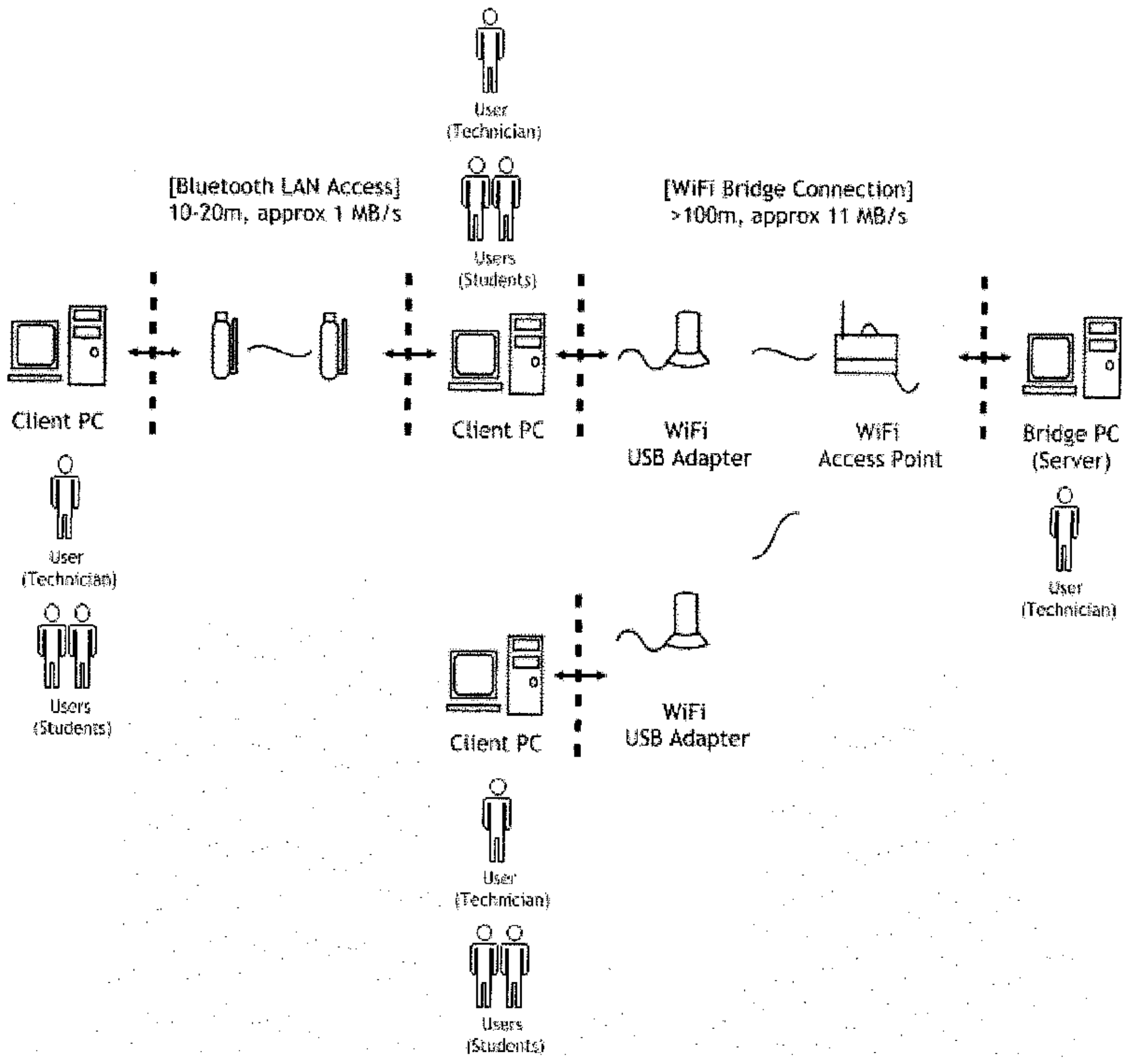
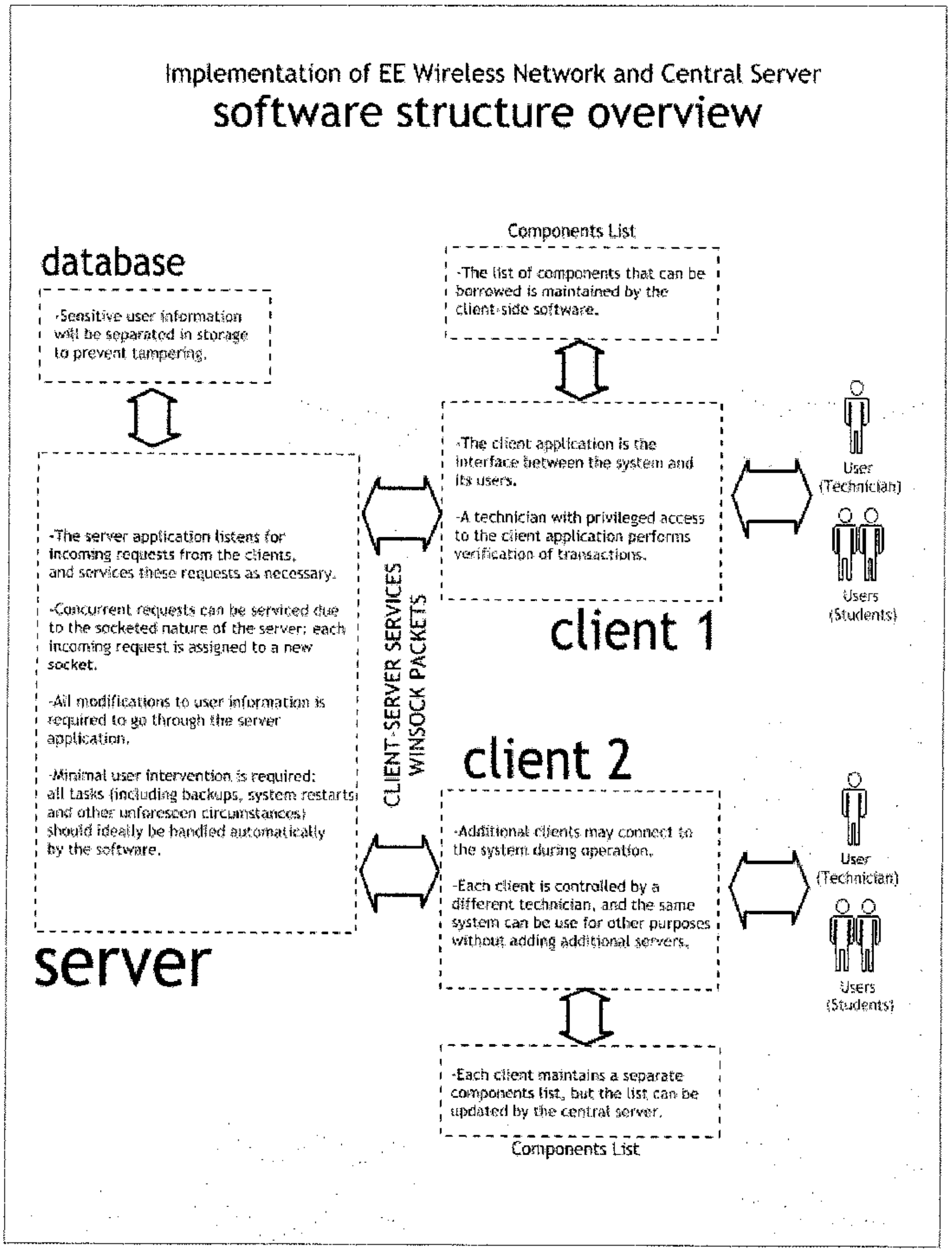


Figure 3.2: Current Hardware Layout

3.3 Software Organization

Figure 3.3 shows the current layout of the software, consisting of separate client and server applications.



-Sensitive user information will be separated in storage to prevent tampering.

-The list of components that can be borrowed is maintained by the client-side software.

-The client application is the interface between the system and its users.

-A technician with privileged access to the client application performs verification of transactions.

-The server application listens for incoming requests from the clients, and services these requests as necessary.

-Concurrent requests can be serviced due to the socketed nature of the server; each incoming request is assigned to a new socket.

-All modifications to user information is required to go through the server application.

-Minimal user intervention is required: all tasks (including backups, system restarts and other unforeseen circumstances) should ideally be handled automatically by the software.

-Additional clients may connect to the system during operation.

-Each client is controlled by a different technician, and the same system can be use for other purposes without adding additional servers.

-Each client maintains a separate components list, but the list can be updated by the central server.

Figure 3.3: Current Software Layout

3.4 Setting up a WiFi Bridge Connection

For the connection between the two blocks, two Cisco Aironet 350 devices are used. The 350 series uses the IEEE 802.11b standard, and the specification sheet for the bridge devices can be found in the appendices. The Aironet 350 is powered using an Ethernet cable instead of a separate power cable, and has its own address on the IP network.

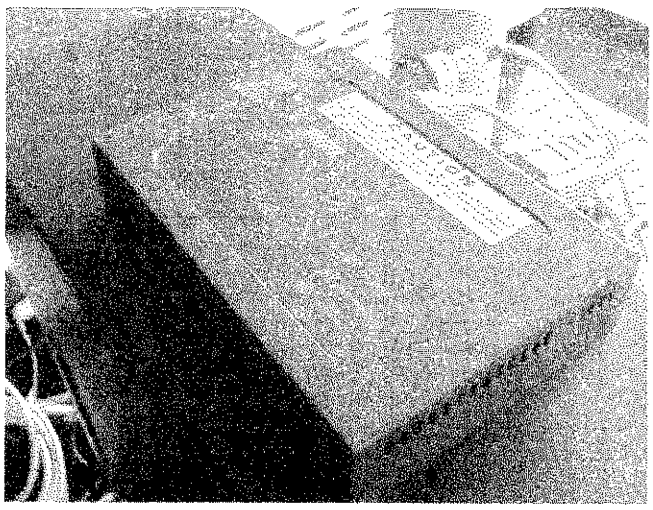


Figure 3.4: The Cisco Aironet 350 overview

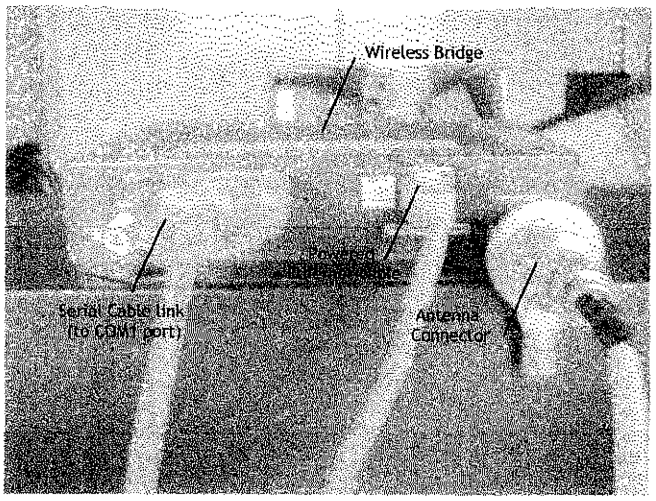
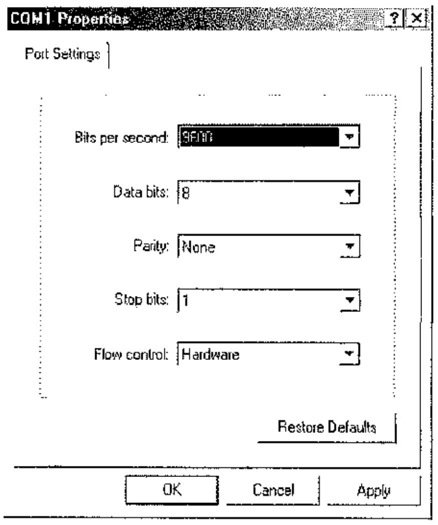


Figure 3.5: Cable connections for the device

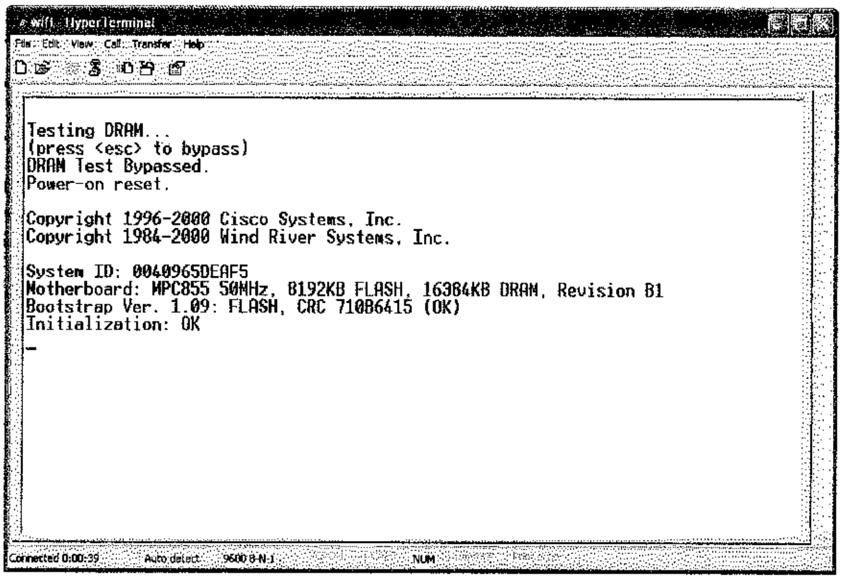
In order to use the device for wireless access, the device has to be configured using the HyperTerminal application in Windows. To do this, the bridge is connected to the PC using a serial cable to the COM1 port, and HyperTerminal is started up. Figure 3.6 shows the settings to use for the connection.



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: Hardware

Figure 3.6: COM1 port settings

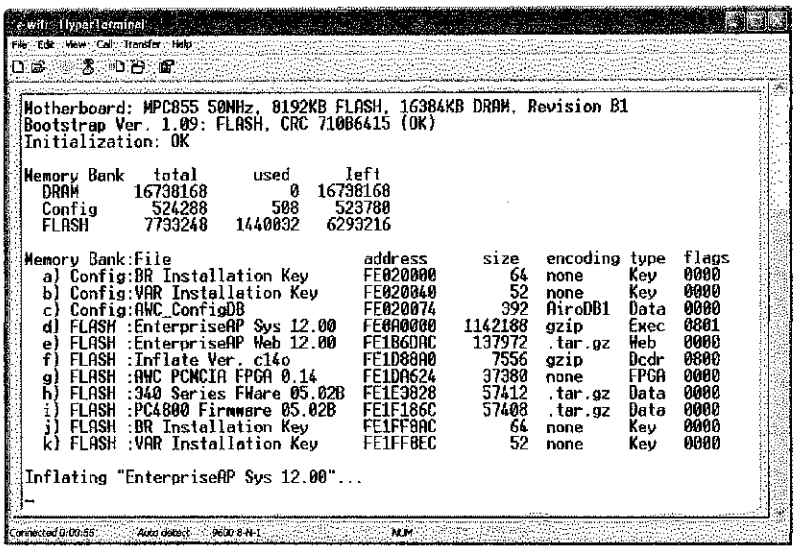
If the settings are configured correctly, the HyperTerminal connection will acknowledge the booting up of the WiFi Bridge as soon as the Ethernet/power cable is connected.



```
Testing DRAM...
(press <esc> to bypass)
DRAM Test Bypassed.
Power-on reset.

Copyright 1996-2000 Cisco Systems, Inc.
Copyright 1984-2000 Wind River Systems, Inc.

System ID: 00409650EAF5
Motherboard: MPC855 50MHz, 8192KB FLASH, 16384KB DRAM, Revision B1
Bootstrap Ver. 1.09: FLASH, CRC 710B6415 (OK)
Initialization: OK
_
```

Connected 0:00:39 Auto detect 9600 8-N-1 NUM

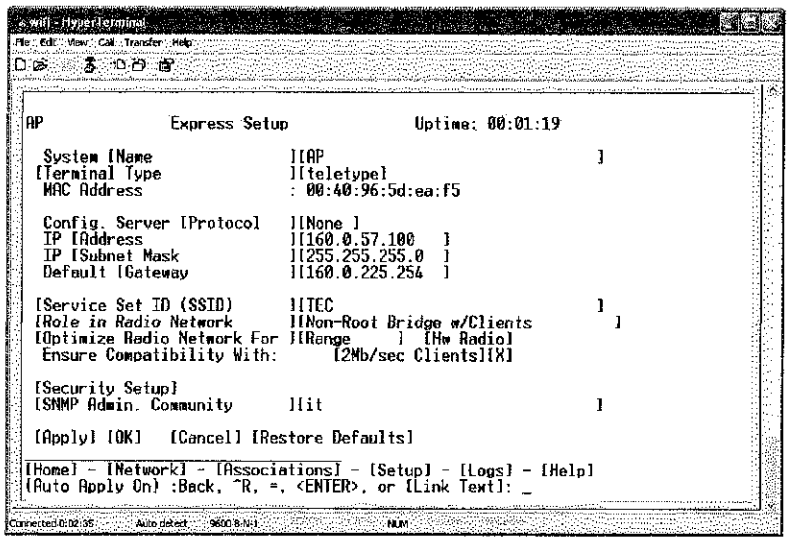
Figure 3.7: HyperTerminal showing Cisco device boot-up sequence



```
Motherboard: MPC855 50MHz, 8192KB FLASH, 16384KB DRAM, Revision B1
Bootstrap Ver. 1.09: FLASH, CRC 710B6415 (OK)
Initialization: OK

Memory Bank   total      used      left
  DRAM      16738168        0  16738168
  Config      524288      508    523780
  FLASH      7733248  1440032   6293216

Memory Bank:File                   address      size  encoding type  flags
  a) Config:BR Installation Key    FE020000       64  none     Key   0000
  b) Config:VAR Installation Key   FE020040       52  none     Key   0000
  c) Config:AWC_ConfigDB           FE020074      392  AiroDB1  Data  0000
  d) FLASH :EnterpriseAP Sys 12.00 FE0A0000  1142188  gzip     Exec  0801
  e) FLASH :EnterpriseAP Web 12.00 FE1B6DAC   137972  .tar.gz  Web   0000
  f) FLASH :Inflate Ver. c14o      FE1D88A0     7556  gzip     Dcdr  0800
  g) FLASH :AWC PCMCIA FPGA 0.14   FE1DA624    37380  none     FPGA  0000
  h) FLASH :340 Series FWare 05.02B FE1E3828   57412  .tar.gz  Data  0000
  i) FLASH :PC4800 Firmware 05.02B FE1F186C    57408  .tar.gz  Data  0000
  j) FLASH :BR Installation Key    FE1FF8AC       64  none     Key   0000
  k) FLASH :VAR Installation Key   FE1FF8EC       52  none     Key   0000

Inflating "EnterpriseAP Sys 12.00"...
_
```

Figure 3.8: Device configuration screen

Once the device has successfully booted up, the Express Setup option allows us to find or change the MAC address, IP address, subnet mask, gateway IP and the role of the device in the radio network.



```
AP                 Express Setup              Uptime: 00:01:19

  System [Name               ][AP                              ]
 [Terminal Type              ][teletype]
  MAC Address                : 00:40:96:5d:ea:f5

  Config. Server [Protocol   ][None ]
  IP [Address                ][160.0.57.100   ]
  IP [Subnet Mask            ][255.255.255.0  ]
  Default [Gateway           ][160.0.225.254  ]

 [Service Set ID (SSID)      ][TEC                             ]
 [Role in Radio Network      ][Non-Root Bridge w/Clients         ]
 [Optimize Radio Network For ][Range       ]  [Hw Radio]
  Ensure Compatibility With:      [2Mb/sec Clients][X]

 [Security Setup]
 [SNMP Admin. Community      ][it                              ]

 [Apply] [OK]   [Cancel] [Restore Defaults]

[Home] - [Network] - [Associations] - [Setup] - [Logs] - [Help]
(Auto Apply On) :Back, ^R, =, <ENTER>, or [Link Text]: _
```

Figure 3.9: Express Setup options

For the purpose of this project, we will need to set up at least two devices. One will function as the root bridge while the other is a non-root bridge that will be connected to further clients. An antenna is then connected to the bridge device to boost the signal.

The antenna used for both devices is Cisco's AIR-ANT1949 wall mounted Yagi-Uda antenna. The specification for this antenna can be found in the appendices of this report. The antenna itself is contained inside a cylindrical shell as shown in Figure 3.10.

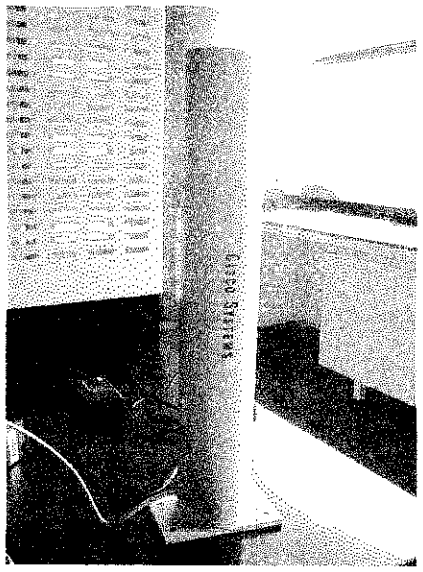


Figure 3.10: Yagi-Uda wall mounted antenna

The antenna's internal layout is shown in the diagram below. Basically, it is a dipole antenna combined with an array of parasitic elements. One element functions as a reflector, while the rest function as directors. The parasitic elements are spaced at equal to a quarter of the signal wavelength apart, but get progressively shorter as it approaches the dipole to direct signals of increasing higher frequencies [6].

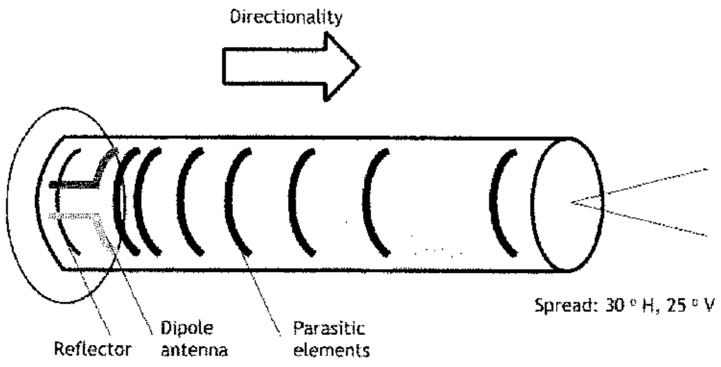


Figure 3.11: Yagi-Uda antenna internal layout

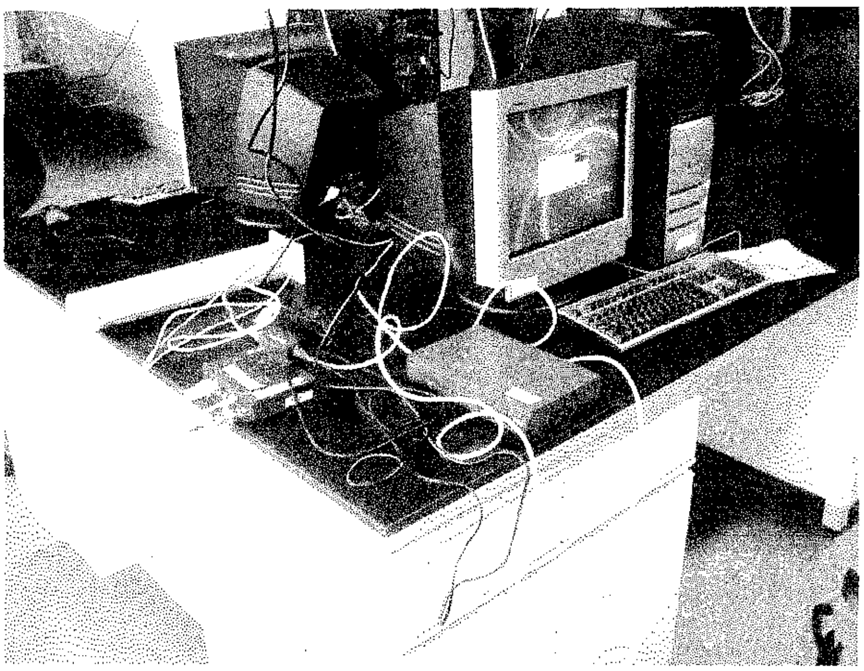


Figure 3.12: WiFi bridge setup (Data Communications Laboratory, Block 23)

3.5 Setting up a Wireless G Network

Setting up a Wireless G network requires configuring a collection of commercial, off-the-shelf hardware. It consists of two main components: the D-Link DWL-2100AP AirPlus XtremeG 108G Wireless Access Point, and several D-Link DWL-G122 AirPlus G USB Adapters.

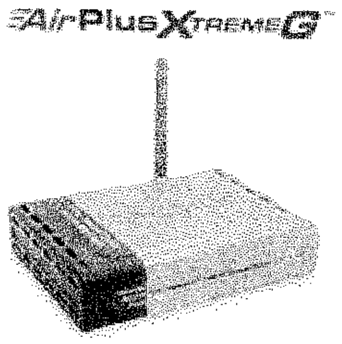


Figure 3.13: D-Link Wireless G Access Point

The main hub of the network will be the access point (pictured in Figure 2.13, previous page) while each client is connected to the Wireless G network using a USB adapter as pictured in Figure 3.14, below.

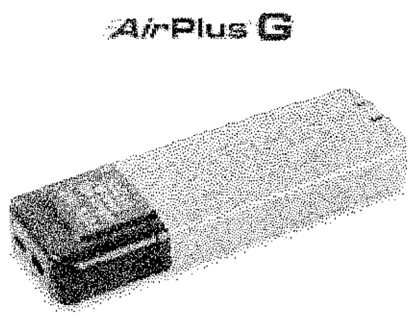
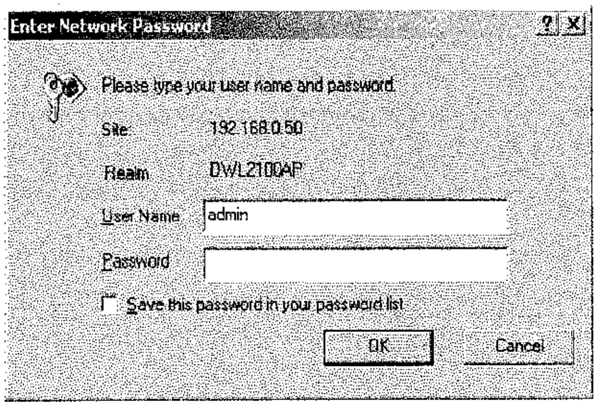


Figure 3.14: Wireless G USB Adapter

The access point is first setup for the wireless network by plugging in the power adapter, then connecting the LAN connector on the access point to a network switch. A configuration PC is then connected to the switch, and will be used to set the configuration for the access point. The default IP for the access point is 192.168.0.50. In order to access the access point, the address http://192.168.0.50/ is entered into a web browser on the configuration PC. A logon popup screen appears, prompting for a username and password (see Figure 3.15). By default, the username is 'admin' and the password field is blank.


Enter Network Password

Please type your user name and password.

Site: 192.168.0.50

Realm: DWL2100AP

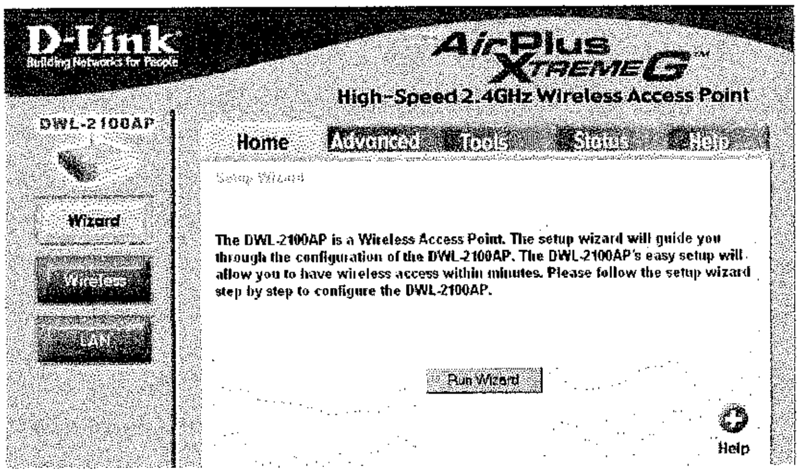
User Name: admin

Password:

Save this password in your password list

Figure 3.15: Access point logon

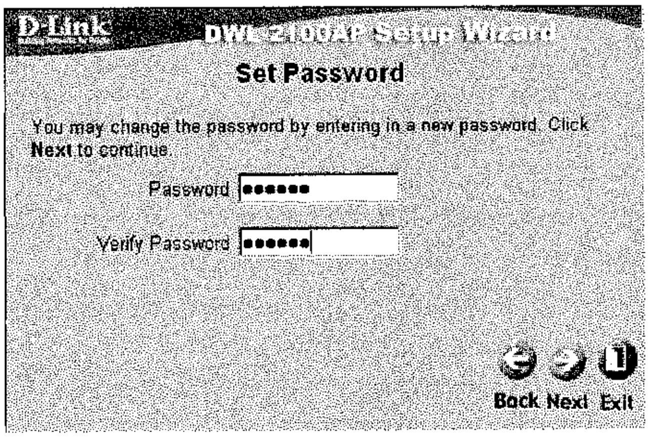
Once the user is verified, the setup wizard will run on the access point. The wizard allows quick configuration of the access point password, the service set identifier and also encryption options for wireless packets handled by the access point. The secure set identifiers (SSID) are special codes, up to 32 characters in length, attached to each packet of the wireless network that allows the system to identify which packets are native to the network.



The DWL-2100AP is a Wireless Access Point. The setup wizard will guide you through the configuration of the DWL-2100AP. The DWL-2100AP's easy setup will allow you to have wireless access within minutes. Please follow the setup wizard step by step to configure the DWL-2100AP.

Figure 3.16: Setup welcome screen

The first setting to be changed is the default access point password. For obvious security reasons, the default blank password should be changed to a new one, as shown in Figure 3.17 below. A strong password should be chosen, consisting of a mixture of uppercase and lowercase letters, as well as special characters and numbers.



You may change the password by entering in a new password. Click **Next** to continue.

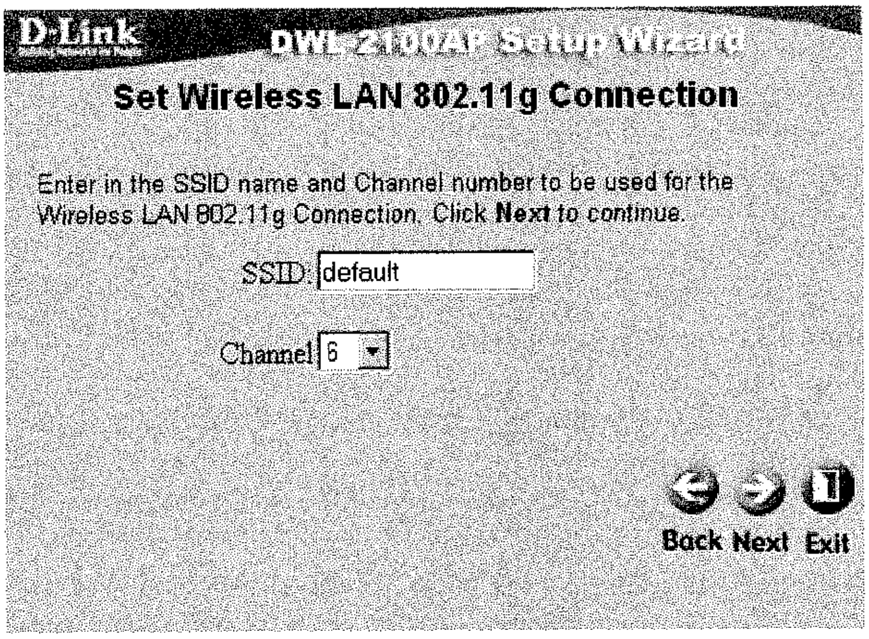
Password ●●●●●●

Verify Password ●●●●●●

Figure 3.17: Password change screen

The second setting to be changed is the wireless connection settings. This is done by setting the broadcast channel and SSID for all packets passing through the access point. The SSID is initially set to a value of 'default', which means that all packets will be tagged with a value decided by the access point. If the user wishes to use a different SSID to tag the packets, then it may be changed here. A common SSID allows the network to be expanded by identifying additional networks as part of the original wireless network.

The broadcast channel is a numeric value that determines which channel on the bandwidth is used to broadcast the packets, ranging from a minimum value of 1 to a maximum value of 11. By default, the value is set to 6. This feature is similar to a channel selection on a walkie-talkie, where only packets on similar channels are able to reach one another clearly. This is useful if interference is an issue, otherwise the default value of 6 can be used.



Set Wireless LAN 802.11g Connection

Enter in the SSID name and Channel number to be used for the Wireless LAN 802.11g Connection. Click **Next** to continue.

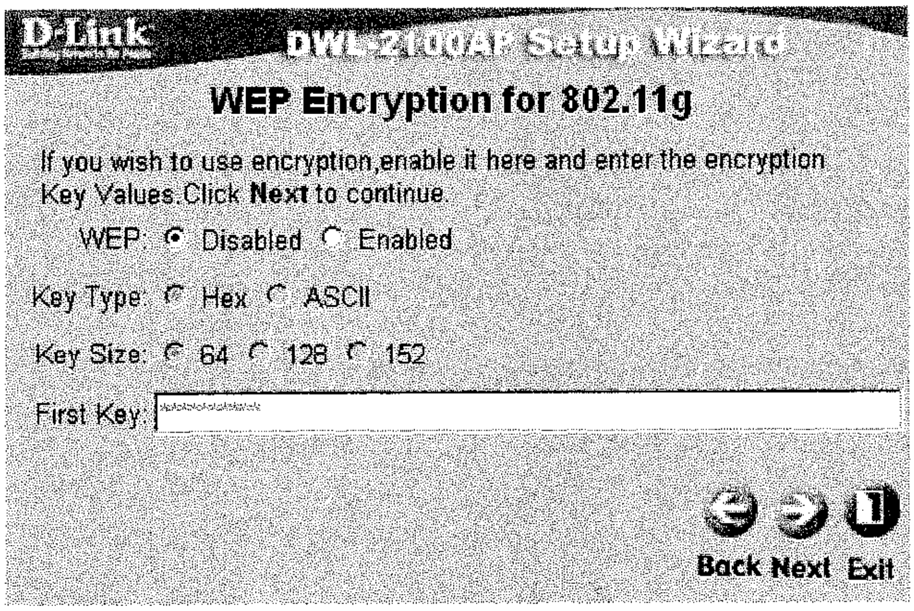
SSID: default

Channel: 6

Figure 3.18: SSID/Channel selection

The third setting is the encryption option for the access point. For more secure wireless communications, encryption of packets is highly recommended. For the D-Link access point, the encryption scheme used is the Wired Equivalent Privacy (WEP) encryption. WEP is performed by seeding the input string with a stream cipher with a cyclic redundancy checksum system. As a stream cipher, it splits the key into a smaller sized initialization vector then uses this vector as the traffic key for all packets passing through the system. Although initially thought to secure a wireless network from external attacks, the WEP scheme has been demonstrated to be mathematically vulnerable to certain types of attacks.

Despite its known weaknesses, a system with WEP encryption is better than a system with no encryption at all. There are several key sizes available for the encryption key, ranging from 64 characters to 152 characters wide. The larger the key size, the less likely the encryption is cracked by radio eavesdroppers. For this project, a key size of 64 characters is enough for security purposes. The key can be either in hexadecimal or the American Standard Code for Information Interchange (ASCII).



WEP Encryption for 802.11g

If you wish to use encryption,enable it here and enter the encryption Key Values.Click **Next** to continue.

WEP: [✓ Disabled] Enabled

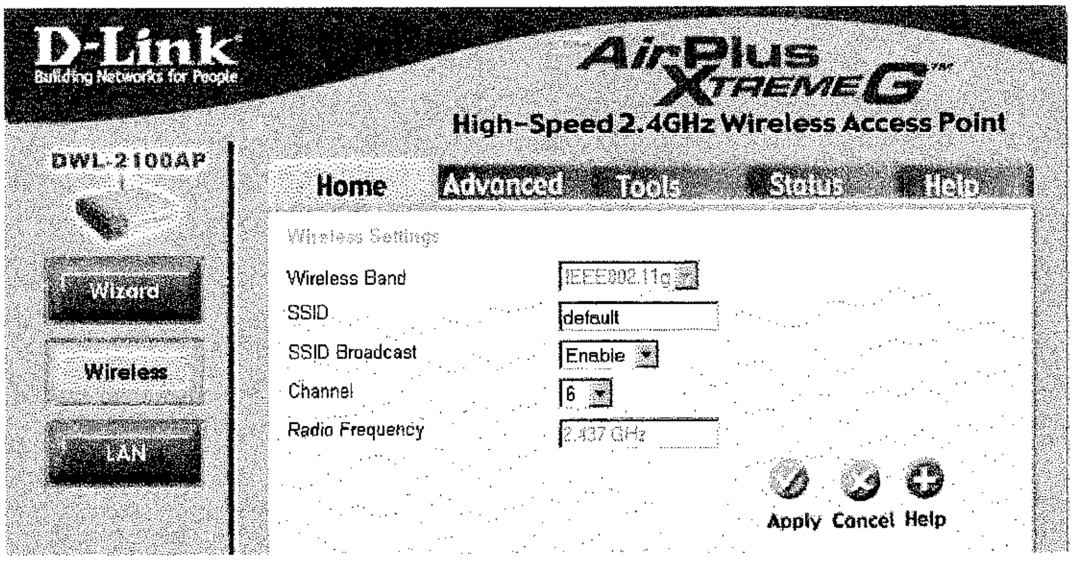
Key Type: [✓ Hex] ASCII

Key Size: [✓ 64] 128 152

First Key: **********

Figure 3.19: WEP encryption options

Once all the options have been set, the access point is operational. The access point control panel now shows all the settings selected, including the wireless band and transmission frequency, broadcast options and packet encryption settings. These can be changed as necessary, but once the access point is set up, it can be disconnected from the network switch and allowed to function as an independent wireless network hub.



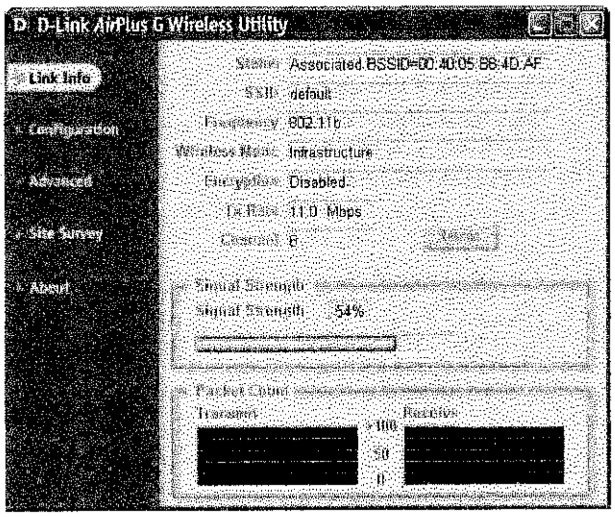
Wireless Settings

Wireless Band	IEEE802.11g
SSID	default
SSID Broadcast	Enable
Channel	6
Radio Frequency	2.437 GHz

Figure 3.20: Access point settings

After the access point is configured, the clients that wish to connect to the wireless network require either a wireless network card or another form of network adapter in order to communicate through radio with the access point. For the purpose of this project, the Wireless G USB Adapter will be used. It is compact, cost-effective and easy to install, thus making it the most suitable solution for the project.

The USB Adapter is plugged into an available USB port on the client computer, and the required drivers are installed. If a wireless network is detected, the Wireless Utility allows the user to select and configure connection settings to the wireless network.



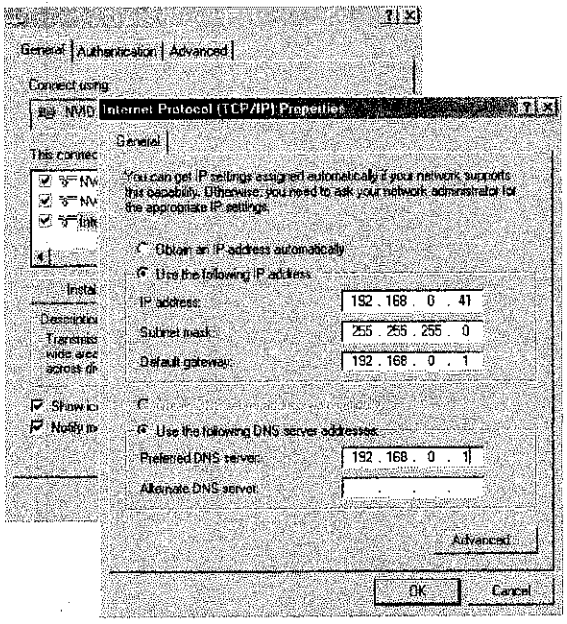
Status: Associated BSSID=00:40:05:B8:4D:AF
SSID: default
Frequency: 802.11b
Wireless Mode: Infrastructure
Encryption: Disabled
TxRate: 11.0 Mbps
Channel: 6

Signal Strength
Signal Strength: 54%

Packet Count
Transmit 100 50 0 Receive

Figure 2.21: Wireless Utility screenshot

The utility that is installed with the drivers shows information such as the current status of the connection, the SSID of the adapter, the frequency used by the adapter, the current wireless connection mode, encryption settings, transmission rate and broadcast channel. It also has a status bar showing the signal strength of the transmission. To complete the setup, the Transmission Control Protocol/Internet Protocol (TCP/IP) settings of the wireless adapter are configured to connect to the wireless access point.


General | Authentication | Advanced
Connect using:
Internet Protocol (TCP/IP) Properties
General
Advanced...
OK
Cancel

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 0 . 41
Subnet mask: 255 . 255 . 255 . 0
Default gateway: 192 . 168 . 0 . 1

[✓ Use the following DNS server addresses:]

Preferred DNS server: 192 . 168 . 0 . 1
Alternate DNS server: . . .

Figure 3.22: Network TCP/IP settings

3.6 Setting up a Bluetooth Connection

There are a few steps required in order to set up a functioning Bluetooth network. First of all, of course, a Bluetooth adapter has to be acquired. These are easily available from electronics stores, with prices ranging from RM35 to RM100. A decent USB Bluetooth dongle can be purchased for RM50.

Figure 3.23: A USB Bluetooth dongle with antenna (100 m/378 ft range)

Most USB dongles come with their own software drivers to get the device running. The standard Bluetooth control software is BlueSoleil, a program developed by IVT Corporation and distributed with commercially available Bluetooth dongles.

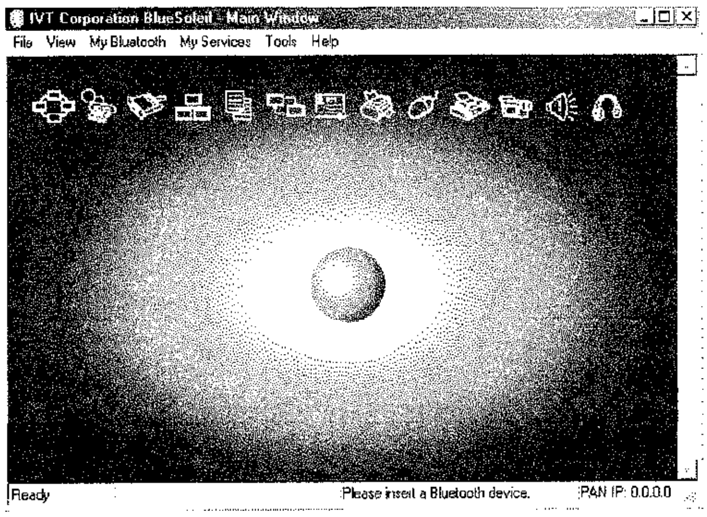

IVT Corporation BlueSoleil - Main Window
File View My Bluetooth My Services Tools Help
Ready
Please insert a Bluetooth device.
PAN IP: 0.0.0.0

Figure 3.24: Screenshot of BlueSoleil 1.6.1.4

A Bluetooth connection functions in several different stages:

1 **Device Inquiry**

In the first stage, a Bluetooth scans the frequency band in order to discover all other devices in range. Devices that are set to 'discoverable' can then be seen by the Bluetooth device running the inquiry. In BlueSoleil, each discovered device will be added to the circular ellipse surrounding the central globe in the interface. Alternatively, if the Bluetooth address is known device can be found directly. A device's Bluetooth address is always a collection of 12 hexadecimal characters divided into 6 groups of 2 characters each, separated by colons, e.g. 01:A2:39:00:45:21.

2 **Services Discovery**

After a device is found, it can be further scanned to find out which services are available. The services available are the types of connections that a destination Bluetooth device will accept.

Service Name	Description
Bluetooth Personal Area Networking Service	Allows a PAN connection to be established between two different Bluetooth devices.
Bluetooth Serial Port Service	Allows the Bluetooth connection to emulate a serial port connection between two different devices.
Bluetooth Local Area Network Access Service	Allows the connecting Bluetooth device to become part of the local area network on the receiving Bluetooth device. In effect, the receiving computer becomes a bridge between two networks.

Bluetooth File Transfer Service	Allows a file to be transferred into a specified folder between two different devices.
Bluetooth Printer Service	Allows a device to send print information to a Bluetooth enabled printer.

Table 3.1: Various Bluetooth services available

3 Device Pairing

Usually, before a service can be accessed on the receiving device, a trusted connection has to be established. In order to preserve the security of the connection, a passkey is required on both the sender and the receiver, and if it matches, the authentication process allows data to be exchanged between the devices. This usually has to be done only once for a given pair of device, although pairings can be edited or deleted at any time. All transmissions can also be encrypted if required, although this is turned off by default to save bandwidth.

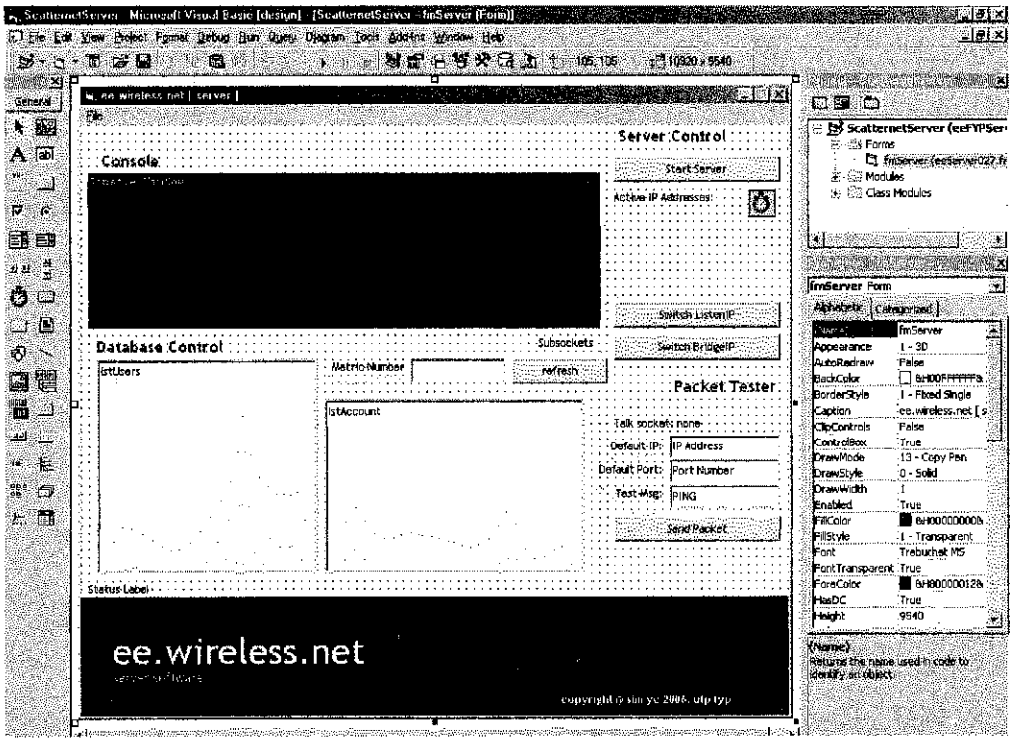
CHAPTER 4

RESULTS & DISCUSSION

In this chapter, the results of the project are discussed in more detail. The final software layout, the various components of the final software, and the results of the software interface are all shown and explained.

4.1 Final Software Layout: Server

The development of the software is done in Microsoft Visual Basic 6.0, and consists of two distinct programs. The most important one is the server software, which is the software equivalent of the wireless network's access point. It handles all incoming requests from clients, processes those requests, then initiates the appropriate actions based on the requests received.



(Name)	fmServer
Appearance	1 - 3D
AutoRedraw	False
BackColor	☐ &H00FFFFFF&
BorderStyle	1 - Fixed Single
Caption	ee.wireless.net [s
ClipControls	False
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	■ &H00000000&
FillStyle	1 - Transparent
Font	Trebuchet MS
FontTransparent	True
ForeColor	■ &H80000012&
HasDC	True
Height	9540

(Name)
Returns the name used in code to identify an object.

Figure 4.1: Visual Basic 6.0 server software development window

All Visual Basic applications consist of two portions: the objects, which are the graphical representation of the program, including the positions and names of buttons, text edit boxes, text display boxes, graphics and so on. The other part is the raw code which determines the behaviour of these objects in response to user input. All the codes used in this project have been included in the related appendix section.

The program for the server actually consists of three separate parts. Each part will be separately described in detail in this report. An overview of the server parts

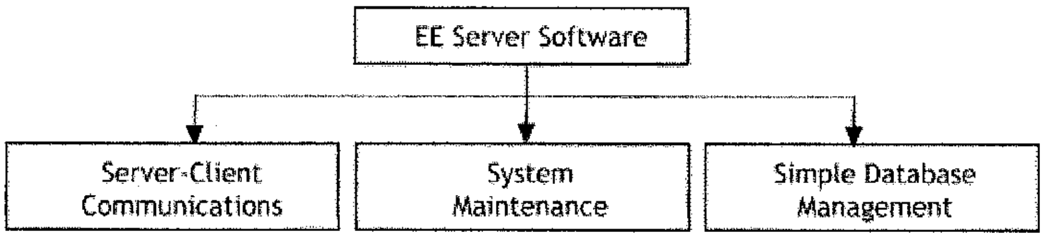


Figure 4.2: Overview of server architecture

1. Server-Client Communications

The codes for this section controls how the server responds to socket events, the structure of the packets sent, as well all the housekeeping on the parts of the software that deals directly with the network.

2. System Maintenance

This section controls the overall system, by processing client requests and servicing them accordingly, checking login authentications, sending status messages to other clients on the network and other background services.

3. Simple Database Management

The server software has database functionalities built in, using random access files in Visual Basic to greatly speed up information retrieval and storage. The database management part deals with how the software communicates with the database files, and returns the correct data to the other parts of the system.

4.1.1 Server Client Communications

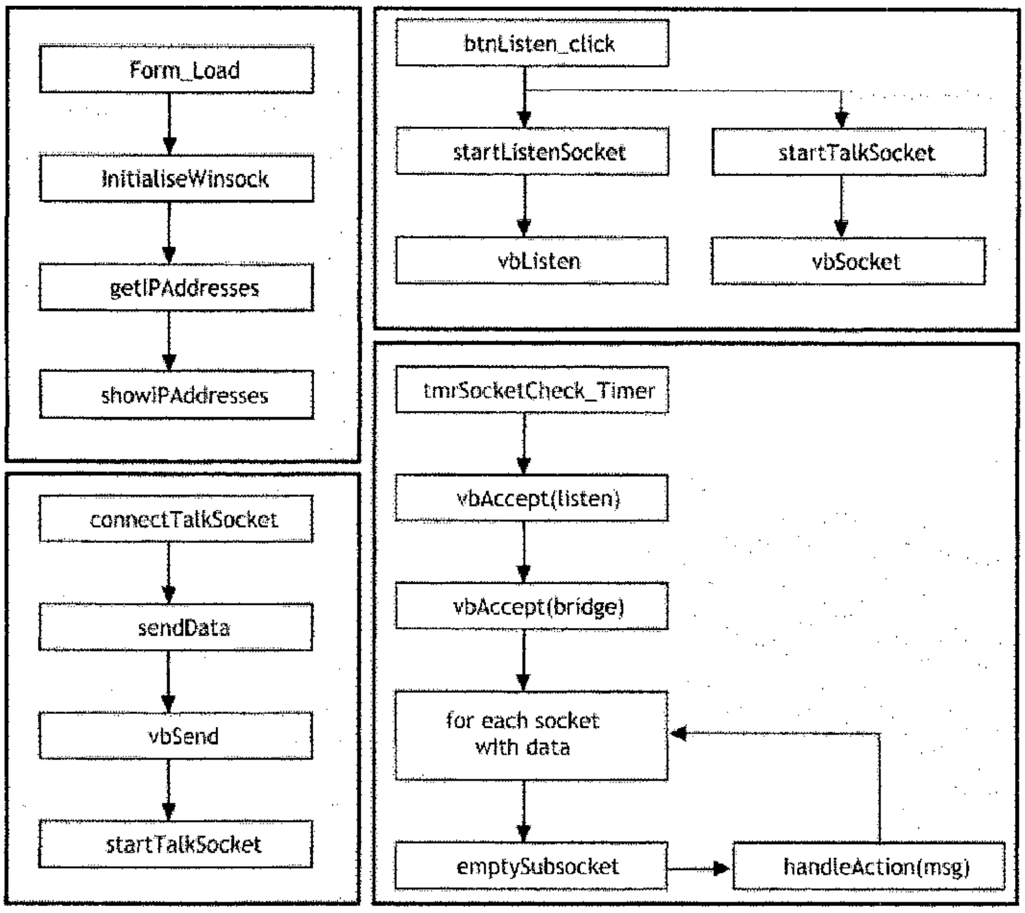


Figure 4.3: Flow-chart for server-client communications

The software codes for server-client communications are identical for both the server and client programs. It relies on Winsock, which is a socket-based communications protocol built into all Windows machines.

When the form is loaded, **Form_Load** initialises the system by calling the function called **InitialiseWinsock**. This triggers the module mdWinSock in the application, which in turn prepares one of Windows' internal dynamic link library (dll) files called "ws2_32.dll" to be used by the program. The Winsock protocol basically opens virtual doors in the system, called sockets, at specific ports of a given IP address. Incoming packets are sent through a socket on the transmitter to a socket on the receiving end of the network, where an additional socket is then created to hold the data contained in the packet sent. Initialising Winsock allows the ws2_32.dll file to handle all the background codes required to keep sockets running. Once initialisation is complete, the function **getIPAddresses** is called. This returns the list of active IP addresses of the computer, which is important so that the user can decide which network will be handling the packets sent by the system. For computers that are located on more than one network (such as being on both a wired network and a wireless network simultaneously), this lets the user decide which IP takes precedence. Once all the valid IP addresses have been returned, the function **showIPAddresses** is called to display the results onto the program's interface.

The server is only started when the button named **btnListen** is clicked. This starts two sockets: a listen socket, for the clients, and a bridge socket, for other server applications. Although the second socket is currently not in use, it allows the system to be expanded to accommodate multiple server programs running side by side. The function **startListenSocket** opens a listening socket on the specified IP address on port 550 then calls the **vbListen** Winsock function to continue listening for changes on the socket. Also, the function **startTalkSocket** opens a transmitting socket on port 552 of the computer, which is marked as a sending socket by the function **vbSocket**.

The listening sockets are checked by a countdown timer called **tmrSocketCheck** which checks for incoming packets approximately 300 ms apart. The timer accepts connection requests on both the listen and bridge sockets, using the **vbAccept** Winsock function then creates one subsocket of data for each request. Then, it loops through all the sockets with data inside, calling **emptySubsocket** on each to extract the source of the packet (including the IP address it originated from) as well as the message contained within the packet. It then continues to call **handleAction** on the message within each socket, and sends this information on to the System Maintenance portion of the program.

When a user wishes to send data, the program simply handles this by running **connectTalkSocket**, **sendData** and **startTalkSocket** in quick succession with the destination IP address, port and the appended message as input arguments.

4.1.2 System Maintenance

The system maintenance portion of the server program handles the client requests. There are currently three possible services:

- Create a new user account

If the user is creating a new account, the system first checks if the user already exists on the system. If not, it passes on the information on the new user to the database management system in order to add a new entry to the user database file.

- Login to an existing account

If the user is tying to log in, the system first determines if the account exists. If it does, then the password required is retrieved by the database system and compared to the password entered by the user. If they are the same, then the server returns an authentication signal, otherwise the return packet informs the client that the login has failed.

- Update an existing account

If the user has made changes to the account, then the server updates the components database based on the changes made. The entire list is sent by the client each time a component change occurs, and the server then reconstructs the component database for a particular user to reflect the changes that have occurred. This is a fast and viable solution because of how the system works: the list of components for each user never shortens, and only lengthens over time, because older components are marked as returned on the list instead of being deleted. This allows the program to accurately predict the minimum length of any new component list, as well as allowing the administrators to keep track of how often the student borrows components.

4.1.3 Simple Database Management

For this project, a customised mini-database is implemented using Random Access Files in Visual Basic. Basically, two data types are defined for the Random Access Files, one for the student record and one for the inventory of each student account. The basic structure for each record is shown below.

```
Private Type StudentRecord
    StudentName As String * 50
    StudentMatric As Long
    StudentIC As String * 15
    AccountActive As Boolean
    StudentIndex As Long
    FirstRecordIndex As Long
End Type
```

The student record consists of six fields: the student's name, the student's matric card number, the student's identification card/passport number, the status of the account, the index of the current student, and the index of the first component in the student's list. All these information are stored in the file named "StudentRecord.db".

```
Private Type AccountRecord
    isFirst As Boolean
    prevRecord As Long
    itemDate As String * 10
    itemStatus As Boolean
    itemName As String * 50
    itemQuantity As Long
    nextRecord As Long
    isLast As Boolean
End Type
```

The component record consists of eight fields: whether the component is the first in the user list, the index of the previous record on the user list, the date the item was returned or borrowed, the status of the item borrowed, the name of the item, the number of items borrowed or returned, the index of the next record in the list and also whether the item is the last in the user's component list. All these information are stored in the file named "AccountRecord.db".

Each student record has a FirstRecordIndex, which functions as a pointer to the first location in the account records. Each account record then points to the next record, similar to a linked list. This enables the entire account to be loaded quickly, as the next record is retrieved as soon as the current record has been processed, since the access times for Random Access Files are the same for any location in the file.

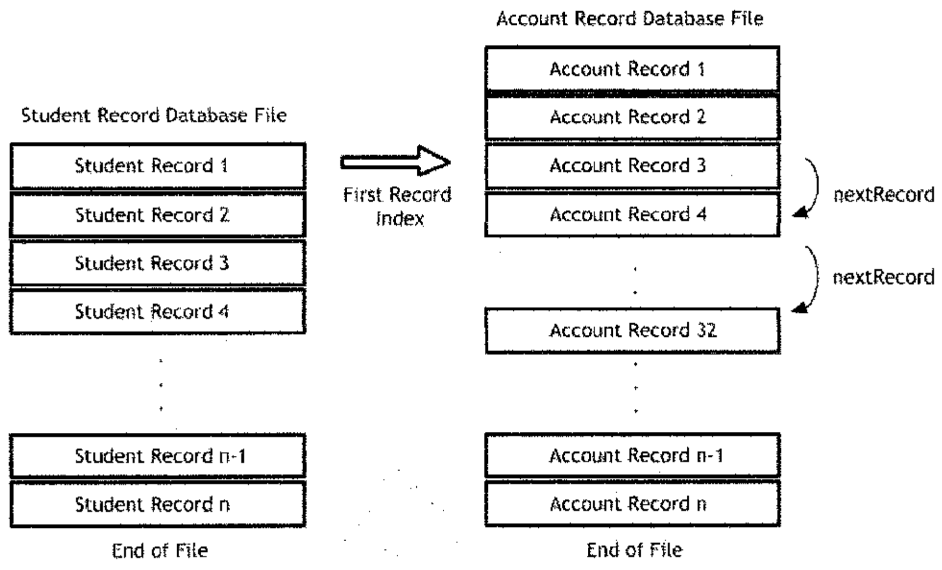
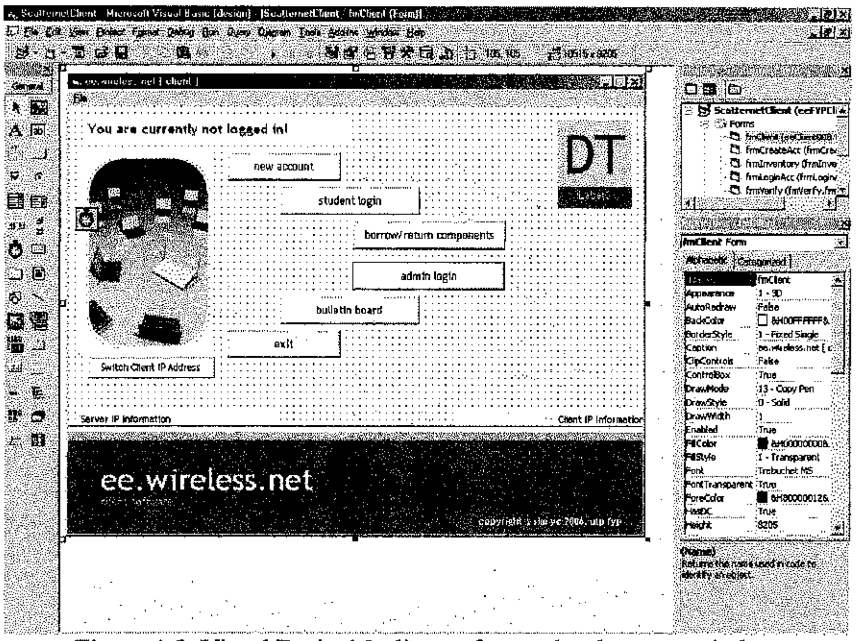


Figure 4.4: Database basic functionality

Once the student is found, the program simply refers to the index of the components, and then quickly returns a list of components associated with the user of interest. This data is then processed by the System Maintenance portion of the program, to be formatted appropriately and then sent back to the client.

4.2 Final Software Layout: Client

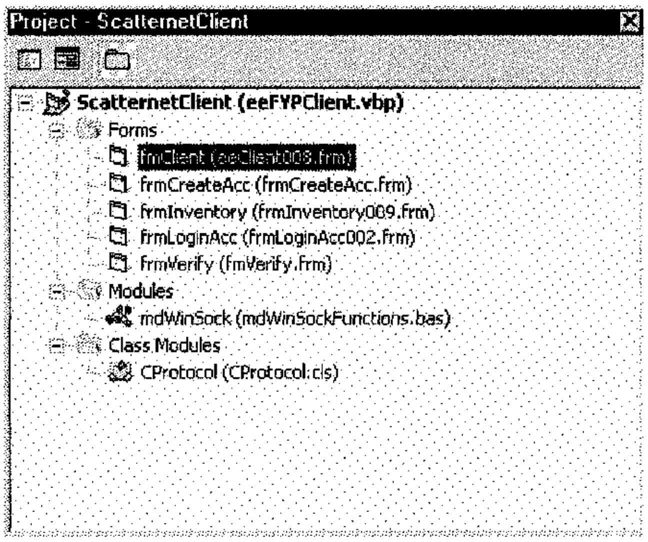


(Name)	frmClient
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H00FFFFFF&
BorderStyle	1 - Fixed Single
Caption	ee.wireless.net [c
ClipControls	False
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	&H00000000&
FillStyle	1 - Transparent
Font	Trebuchet MS
FontTransparent	True
ForeColor	&H80000012&
HasDC	True
Height	8205

(Name)
Returns the name used in code to identify an object.

Figure 4.5: Visual Basic 6.0 client software development window

The client was developed with a different objective compared to the server software. Where the server was meant to be designed to have minimal user intervention, the client software is designed for maximum user interaction, since it will be the most commonly used interface between the users and the system. Therefore, the client consists of a series of forms, each one serving a different function.



Project - ScatternetClient

```
ScatternetClient (eeFYPClient.vbp)
  Forms
    frmClient (eeClient008.frm)
    frmCreateAcc (frmCreateAcc.frm)
    frmInventory (frmInventory009.frm)
    frmLoginAcc (frmLoginAcc002.frm)
    FrmVerify (FmVerify.Frm)
  Modules
    mdWinSock (mdWinSockFunctions.bas)
  Class Modules
    CProtocol (CProtocol.cls)
```

Figure 4.6: List of client forms

- fmClient

The main client form initialises the functions required for communication with the server, then presents the user with a choice of buttons for navigation. It also displays important data such as the current logged in user, the status of requests as well as the date and time.

- frmCreateAccount

The account creation form allows the user to input the data required for the new account, which will be sent to the server for verification.

- frmInventory

The inventory management form has the most functions. When loaded, it requests the complete list of components associated with the logged in user from the server, then retrieves a list of components for the user to be displayed in drop-down lists categorized by class. The user is then allowed to select the number of components to be borrowed, which are added to a shopping basket in the form of a Visual Basic listview. Once the selection is finalised, the items are added to the user account and the entire list is sent to the server for an account update when the user returns to the main menu.

- frmLoginAccount

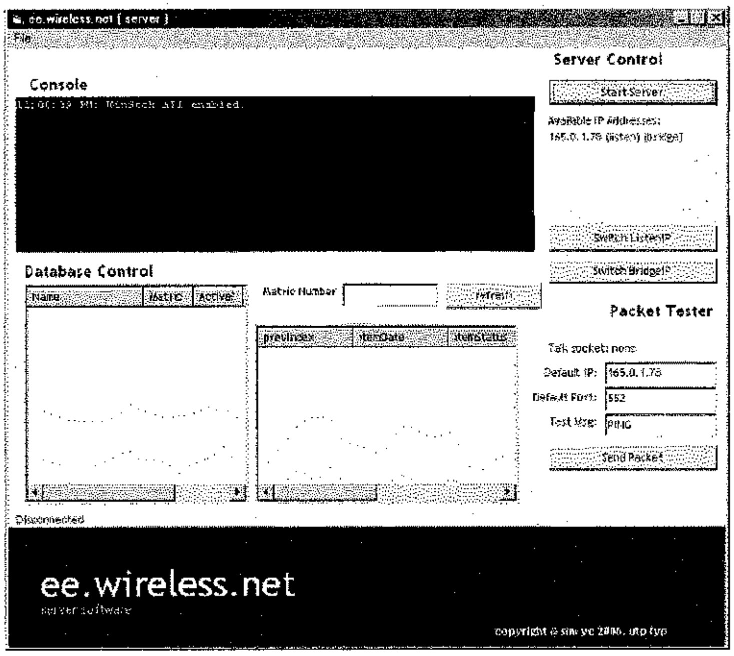
The account login form is a simple one: the user inputs the user name and password, then awaits verification from the server.

- frmVerify

The verification form requires an authentication code from a technician in order to complete sensitive requests.

Most of the functions in the project have been completed. The server and client applications were completely coded in Visual Basic 6, and will run on all Windows machines. Basically, the applications make use of the WinSock API in Windows to transfer data between different locations. A socket is opened at the receiver, and data is placed within a dynamically assigned socket. Each application periodically searches for active sockets, and resolve them as necessary based on the type of data received.

A screenshot of the functioning server application is shown in **Figure 4.7**. It consists of a console that records the most recent actions performed, a server control box to stop or start the server, a list of active sub-sockets, and a packet tester to send test data to a specified IP address. Most of the processing here is done behind the scenes, since the server should involve as little user intervention as possible.



```
11:00:59 PM: WinSock API enabled.
```

Available IP Addresses:
165.0.1.78 (listen) (bridge)

Talk socket: none
Default IP: 165.0.1.78
Default Port: 552
Test Msg: PING

copyright © sim yc 2006. utp fyp

Figure 4.7: Screenshot of ScatterNet Server

The client application is the main concern for this project. It should be user-friendly while robust, and quick to process data while secure. Because of these requirements, a lot of attention has gone into the design of the client software in order to make it intuitive yet flexible enough to deal with the many different functions it has to perform. The client interface has been designed to use well labelled, easily understandable buttons, as well as helpful text prompts that show the user important information such as the current server, the client IP address, as well as the login status.

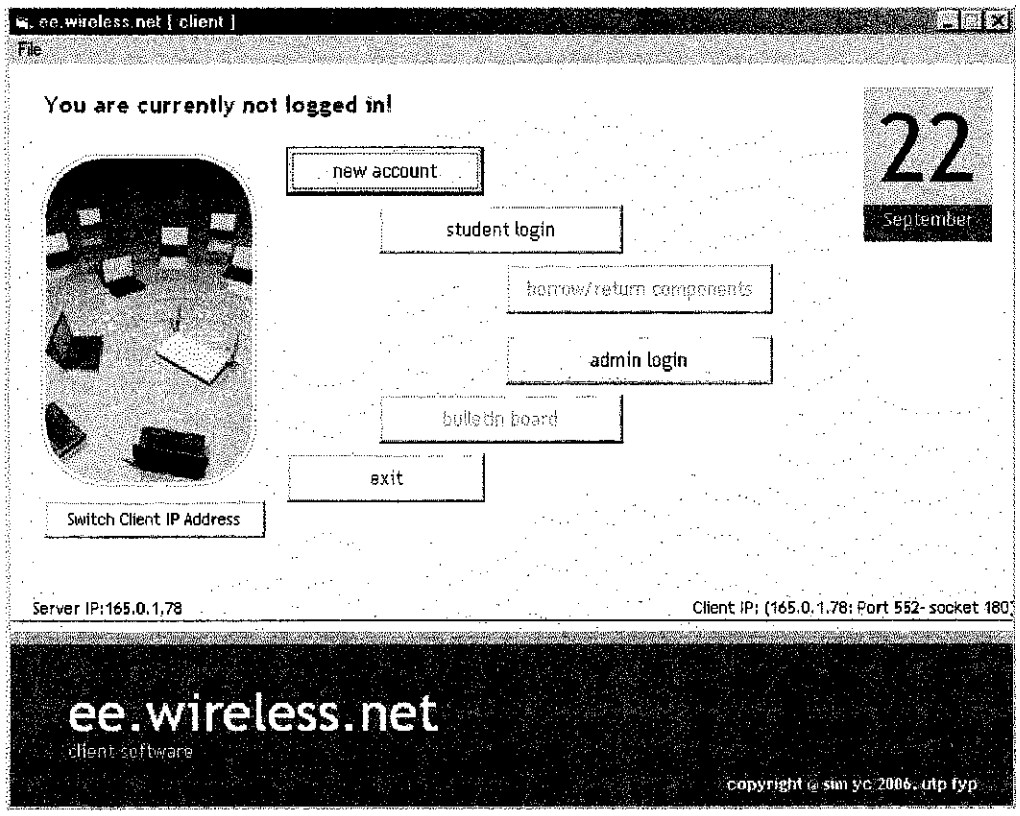


Figure 4.8: Screenshot of ScatterNet Client

Students are required to log onto the application in order to use it, while technicians will be able to access a higher level of commands and options using a special key provided. For now, the login ID of a student is his matric card number, while the password is the student's identity card (IC) number, which is printed on every matric card. This ensures that the student does not have to remember additional passwords in order to access the system. International students may use their passport number instead. An example of the login process is shown in Figure 4.9.

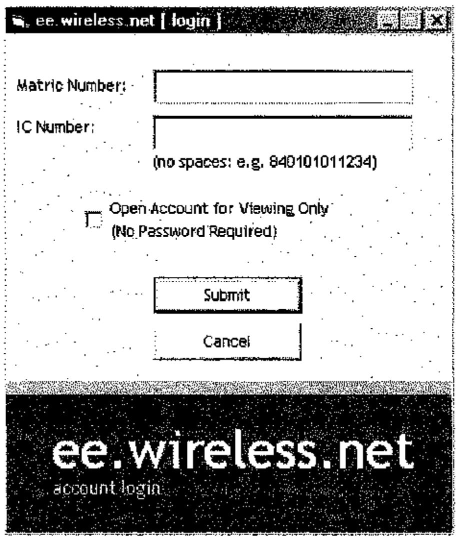
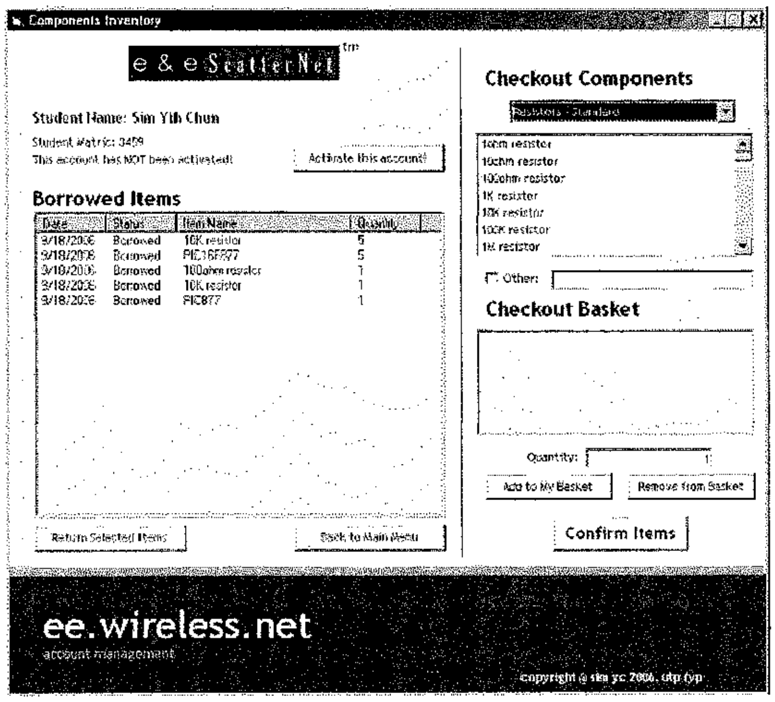


Figure 4.9: Screenshot of ScatterNet Client login interface

The inventory interface as seen in **Figure 34** is designed to be as intuitive as possible. A list of available components is shown on the right, and selected components are then added to a Checkout Basket. Once the user is sure, the items are added to the user account. Returned items are then marked as returned (together with the date it was returned), subject to the technician's approval.



Student Name: Sim Yih Chun
Student Matric: 3459
This account has NOT been activated!

Borrowed Items

Date	Status	Item Name	Quantity
9/18/2006	Borrowed	10K resistor	5
9/18/2006	Borrowed	PIC16F877	5
9/18/2006	Borrowed	100ohm resistor	1
9/18/2006	Borrowed	10K resistor	1
9/18/2006	Borrowed	PIC877	1

1ohm resistor
10ohm resistor
100ohm resistor
1K resistor
10K resistor
100K resistor
1M resistor

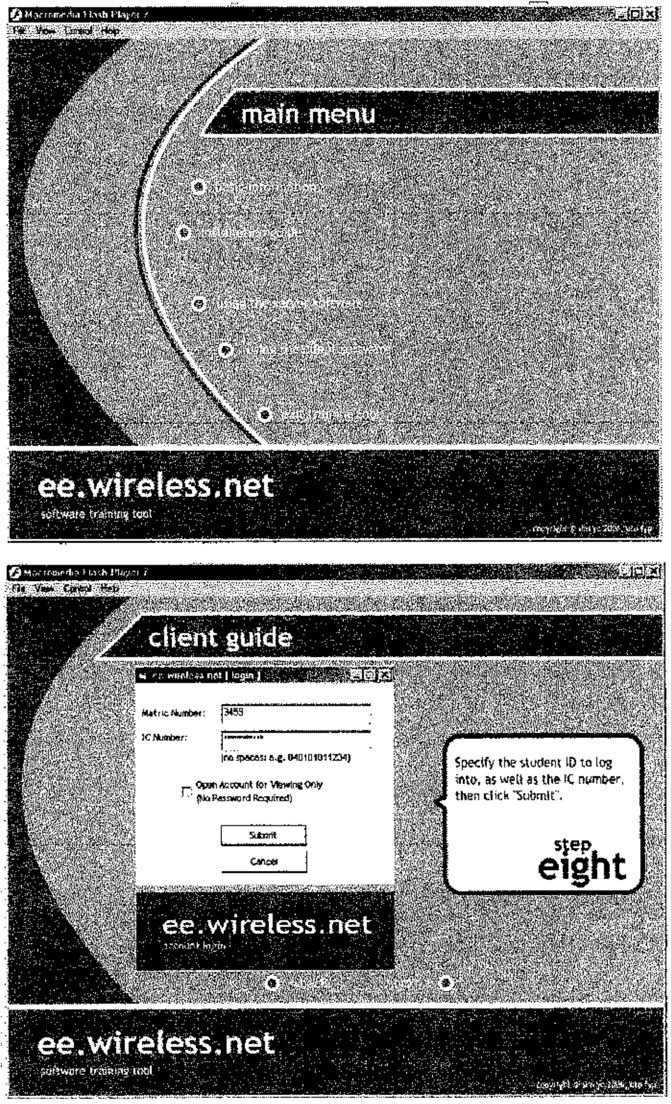
Quantity: 1

Checkout Basket

Figure 4.10: Screenshot of inventory management interface

4.3 Software Training Tool

In order to quickly familiarise the technicians and students to the new system, a software training tool was also developed as part of this project. It was designed and programmed using Macromedia Flash MX 2004. It consists of an interactive series of images that forms a comprehensive, step-by-step guide on using the system. It is hoped that this training tool will make it much easier to transition from the old system to the new one.



Matric Number: 3459
IC Number:
(no spaces: e.g. 040101011234)
Open Account for Viewing Only (No Password Required)

Specify the student ID to log into, as well as the IC number, then click "Submit".

Figure 4.11: Screenshots of software training tool

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

The final chapter outlines the conclusion that was obtained at the end of the project, followed by a few recommendations for further improvements to the system in the future.

5.1 Conclusion

By the end of the project, both the main objectives stated had been successfully achieved. Firstly, a wireless network has been successfully set up in the designated blocks, providing wireless capabilities to the Electrical and Electronics faculty. This was done by using a combination of both WiFi and Bluetooth network components, with a WiFi access point located in Block 22 of the faculty, WiFi adapters in the central store and laboratories, as well as Bluetooth dongles to further expand the range of the network within the same rooms.

The software developed throughout this project brings more practical applications to the network. A WinSock server-client system enables the different components within the network to communicate with one another, while the internal database stores the information required by the inventory management system. A user-friendly, intuitively designed inventory borrowing interface allows the user to interact with the system, while additional security features were implemented to allow the technicians to act as moderators.

In conclusion, it is hoped that the full implementation of this system in the future to replace the existing paper-based system will greatly increase productivity and reliability in the Electrical and Electronics Department's stores, and with further improvements, the wireless network can be expanded to be even more useful.

5.2 Recommendation

For future improvements to the project, additional features can be added to take advantage of the Bluetooth functionality, perhaps by interfacing the inventory system with hand-phones and mobile technology. With the widespread penetration of Bluetooh capabilities in the mobile communication industry, there is great potential in future applications of having Bluetooth connectivity in the network.

Also, a more efficient sorting algorithm for student data can be used in order to speed up the internal database even faster. The current algorithm is sufficiently fast for the existing students, but a better algorithm will allow the system to function fast enough to be extended even further into the future.

Another recommendation is the addition of further functionalities into the system, such as utilising the wireless network capabilities to record student attendance, or to deliver academic material such as notes and important notices. In order for this to be possible, the wireless network has to be expanded to as many rooms as possible within the faculty. The existence of an intelligent hotspot on campus will be very useful to the students.

REFERENCES

[1] Wikipedia, "*Wi-Fi*", 15 April 2006, http://en.wikipedia.org/wiki/Wifi

[2] Bluetooth SIG, *Bluetooth Special Interest Group Launches Bluetooth Core Specification Version 2.0 + Enhanced Data Rate*, (November 8, 2004), press release.

[3] Daniel Minoli, *Hotspot Networks: WiFi for Public Access Locations*, McGraw Hill, 2003.

[4] Cycom Sdn Bhd, *Cycom Sales & Services Sdn Bhd updated Price Quotes*, 1 April 2006, http://www.lowyat.net/v2/pricelist/cycom.pdf

[5] Bob Quinn and David Shute, *Windows Sockets Network Programming,* Addison-Wesley, 2000.

[6] Wikipedia, *Yagi-Uda antenna*, 2006, http://en.wikipedia.org/wiki/Yagi_antenna

Appendix A: Server Software Source Code

```
'-----------------------------------------------------
'Form     : fmServer
'Function : collects information from clients and does
'           all central processing functions
'-----------------------------------------------------
'
'By: Sim Yih Chun
'
'-----------------------------------------------------

Option Explicit

'Prepare array for the last 10 console events
Dim consoleEvents(10) As String

'Prepare array for system's ip addresses
'intValidIp stores the number of valid ip addresses found
'strActiveIp is a global variable that stores the active IP
address
Dim arrIp(6) As String
Dim intValidIp As Integer
Dim intListenIP As Integer
Dim intBridgeIP As Integer
Dim intMaxSockets As Integer
Dim bolServerStarted As Boolean

'Saves the socket handles as global variables
Dim lngListenSocketHandle As Long
Dim lngBridgeSocketHandle As Long
Dim lngTalkSocketHandle As Long
Dim arrSubsocketHandles(16) As Long

'Return value handles in main procedures
Dim lngRetBridgeSocket As Long
Dim lngRetListenSocket As Long
Dim lngRetTalkSocket As Long

'Stores the port numbers for the bridge/listen IPs
Dim lngBridgePort As Long
Dim lngListenPort As Long

'Define data type for the student record database
Private Type StudentRecord
    StudentName As String * 50
    StudentMatric As Long
    StudentIC As String * 15
    AccountActive As Boolean
    StudentIndex As Long
    FirstRecordIndex As Long
End Type

Dim StudentData As StudentRecord
Dim AllStudentData As StudentRecord
Dim StudentFile&
Dim StudentRecordLength&
Dim totalStudentRecords&

Dim accountExists As Boolean

'Define data type for the inventory database
Private Type AccountRecord
    isFirst As Boolean
    prevRecord As Long
    itemDate As String * 10
    itemStatus As Boolean
    itemName As String * 50
    itemQuantity As Long
    nextRecord As Long
    isLast As Boolean
End Type

Dim AccountData As AccountRecord
Dim AllAccountData As AccountRecord
Dim AccountFile&
Dim AccountRecordLength&
Dim totalAccountRecords&

Private Sub btnListenIP_Click()

    'Switch the active listen IP to the next available IP address

    If bolServerStarted = False Then

        intListenIP = intListenIP + 1

        If intListenIP > intValidIp Then

            intListenIP = 1

        End If

        showIPAddresses

        updateConsole     ("ListenIP     changed     to      "      +
arrIp(intListenIP))

    Else

        updateConsole ("Error: Server is already running.")

    End If

End Sub

Private Sub btnBridgeIP_Click()

    'Switch the active bridge IP to the next available IP address
    If bolServerStarted = False Then

        intBridgeIP = intBridgeIP + 1

        If intBridgeIP > intValidIp Then

            intBridgeIP = 1

        End If

        showIPAddresses

        updateConsole    ("BridgeIP    changed    to    "    +
arrIp(intBridgeIP))

    Else

        updateConsole ("Error: Server is already running.")

    End If

End Sub

Private Sub btnListen_Click()

    'triggers the opening/closing of the listen socket
    If bolServerStarted = False Then

        startListenSocket
        startTalkSocket

    Else

        stopAllSockets

    End If

End Sub

Private Sub btnRefreshList_Click()

    Dim retValue As Integer
    retValue = showAllUsers()

End Sub

Private Sub btnSend_Click()

    'sends a test packet to the specified ip/port
    lngRetTalkSocket    =    connectTalkSocket(lngTalkSocketHandle,
txtPingIP.Text, CInt(txtPingPort.Text))
    sendData (txtMessage.Text)

End Sub

Private Sub Form_Load()
```

```
    'initial values are set.
    txtConsole.Caption = ""
    txtStatus.Caption = "Disconnected"
    lngListenSocketHandle = 0
    bolServerStarted = False
    intMaxSockets = 8
    txtSubsockets.Caption = ""

    'set the bridge/listen port numbers
    lngListenPort = 550
    lngBridgePort = 551

    'value returned by the InitializeWinsock function
    Dim lngRetValue As Long

    'initialize the Winsock service
    lngRetValue = mdWinSock.InitializeWinsock(SOCKET_VERSION_22)

    If lngRetValue = 0 Then

        'if the Winsock service was initialized
        'successfully, initialize the controls
        updateConsole ("WinSock API enabled.")

    Else

        'if the Winsock service was not initialized
        'successfully, show the error
        updateConsole ("Unable to initialise WinSock API!")

    End If

    getIPAddresses

    'prepare the database files: open for random access
    StudentFile = FreeFile
    StudentRecordLength = Len(StudentData)
    Open  "StudentRecord.db"  For  Random  As  StudentFile  Len  =
StudentRecordLength
    totalStudentRecords = LOF(StudentFile) \ StudentRecordLength

    AccountFile = FreeFile
    AccountRecordLength = Len(AccountData)
    Open  "AccountRecord.db"  For  Random  As  AccountFile  Len  =
AccountRecordLength
    totalAccountRecords = LOF(AccountFile) \ AccountRecordLength

End Sub


Private Sub Form_Unload(Cancel As Integer)

    'shut down all files and sockets
    Close StudentFile, AccountFile
    Call WSACleanup
```

```
End Sub

Private Sub menuExit_Click()

    Unload Me

End Sub

Private Function updateConsole(consoleNewEvent As String)

    'updates the console's 10 event buffer
    Dim intCount As Integer

    'Shifts down the last 9 recorded events
    For intCount = 1 To 9
        consoleEvents(intCount) = consoleEvents(intCount + 1)
    Next intCount

    'Records the new event
    consoleEvents(10) = "" + CStr(Time) + ": " + consoleNewEvent +
vbCrLf

    'Reinitialise events and display
    'Clear the console and reload all events
    intCount = 1
    txtConsole.Caption = ""

    For intCount = 1 To 10
        txtConsole.Caption      =      txtConsole.Caption      +
consoleEvents(intCount)
    Next intCount

End Function

Private Function startListenSocket()

    'Update status display and console
    txtStatus.Caption = "Starting..."
    btnListen.Caption = "Stop Server"
    updateConsole ("Starting up server: opening sockets...")
    bolServerStarted = True

    'This is the handle of the socket to be created
    Dim lngSocket As Long

    Dim lngAddressFamily As Long
    Dim lngSocketType As Long
    Dim lngProtocol As Long

    lngAddressFamily = 2     '2 = AF_INET: inter-network address
family
    lngSocketType = 1       '1 = SOCK_STREAM: socket streaming
    lngProtocol = 6            '6 = IPPROTO_TCP: transfer control
protocol
```

```
    'Create the sockets: 1 bridge socket between servers and 1
listen socket
    lngSocket          =        mdWinSock.vbSocket(lngAddressFamily,
lngSocketType, lngProtocol)
    lngListenSocketHandle = lngSocket

    lngSocket          =        mdWinSock.vbSocket(lngAddressFamily,
lngSocketType, lngProtocol)
    lngBridgeSocketHandle = lngSocket

    If lngSocket = INVALID_SOCKET Then

        'If the function has returned the INVALID_SOCKET
        'value the socket was not created.
        txtStatus.Caption = "Disconnected"
        updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    Else

        'If a new socket was created successfully, add the
        'listview's item for that socket
        txtStatus.Caption = "ListenIP: (" + arrIp(intListenIP) +
"-sock" & _
                        CStr(lngListenSocketHandle) + ")" & _
                            "/ BridgeIP: (" + arrIp(intBridgeIP) +
"-sock" & _
                        CStr(lngBridgeSocketHandle) + ")"

        updateConsole ("Listen socket opened: " & _
                    CStr(lngListenSocketHandle))
        lngRetListenSocket = bindOneSocket(lngListenSocketHandle,
arrIp(intListenIP), lngListenPort)
        lngRetListenSocket                                          =
listenOneSocket(lngListenSocketHandle)

        updateConsole ("Bridge socket opened: " & _
                    CStr(lngBridgeSocketHandle))
        lngRetBridgeSocket = bindOneSocket(lngBridgeSocketHandle,
arrIp(intBridgeIP), lngBridgePort)
        lngRetBridgeSocket                                          =
listenOneSocket(lngBridgeSocketHandle)

    End If

End Function

Private Function informClient(IPAddress As String, strMsg As
String)

    lngRetTalkSocket    =    connectTalkSocket(lngTalkSocketHandle,
IPAddress, CInt(txtPingPort.Text))
    sendData (strMsg)

End Function
```

```
Private Function stopAllSockets()

    updateConsole ("Shutting down server: closing all sockets...")

    'shuts down the listen server
    'close the listening socket handle
    Call closesocket(lngListenSocketHandle)
    updateConsole      ("Listen      socket      closed:      "      +
CStr(lngListenSocketHandle))
    lngListenSocketHandle = 0

    Call closesocket(lngBridgeSocketHandle)
    updateConsole      ("Bridge      socket      closed:      "      +
CStr(lngBridgeSocketHandle))
    lngBridgeSocketHandle = 0

    Call closesocket(lngTalkSocketHandle)
    updateConsole      ("Talk      socket      closed:      "      +
CStr(lngTalkSocketHandle))
    lngTalkSocketHandle = 0
    txtTalkSocket.Caption = "Talk socket: none"

    Dim intCount As Integer

    For intCount = 1 To 16

        If arrSubsocketHandles(intCount) > 0 Then

            Call closesocket(arrSubsocketHandles(intCount))
            updateConsole      ("Subsocket      closed:      "      +
CStr(arrSubsocketHandles(intCount)))
            arrSubsocketHandles(intCount) = 0

        End If

    Next intCount

    refreshSubsockets

    txtStatus.Caption = "Disconnected"
    btnListen.Caption = "Start Server"

    updateConsole ("Server disconnected.")

    bolServerStarted = False

End Function


Private  Sub  Form_QueryUnload(Cancel  As  Integer,  UnloadMode  As
Integer)

    If Not lngListenSocketHandle = 0 Then

        updateConsole ("Error: " + arrIp(intListenIP) + ": socket
" & _
                        CStr(lngListenSocketHandle) + " still open.")
        Cancel = 1

    End If

    If Not lngBridgeSocketHandle = 0 Then

        updateConsole ("Error: " + arrIp(intBridgeIP) + ": socket
" & _
                        CStr(lngBridgeSocketHandle) + " still open.")
        Cancel = 1

    End If

End Sub

Private Function getIPAddresses()

    'pointer to HOSTENT structure returned by
    'the gethostbyname function
    Dim lngPtrToHOSTENT As Long

    'structure which stores all the host info
    Dim udtHostent      As HOSTENT

    'pointer to the IP address' list
    Dim lngPtrToIP      As Long

    'byte array that contains elemets of an IP address
    Dim arrIpAddress()  As Byte

    'result IP address string to add into the ListBox
    Dim strIpAddress    As String

    'buffer string to receive the local system host name
    Dim strHostName As String * 256
    'value returned by the gethostname function
    Dim lngRetVal As Long
    Dim i As Integer

    'Get the local host name
    lngRetVal = gethostname(strHostName, 256)

    If lngRetVal = SOCKET_ERROR Then
        MsgBox   "Can't   resolve   the   local   host   address",
vbExclamation
        Exit Function
    End If

    'Call the gethostbyname Winsock API function
    'to get pointer to the HOSTENT structure
    lngPtrToHOSTENT   =   gethostbyname(Left(strHostName,   InStr(1,
strHostName, Chr(0)) - 1))

    'Check the lngPtrToHOSTENT value
    If lngPtrToHOSTENT = 0 Then

        'If the gethostbyname function has returned 0
        'the function execution is failed.

        updateConsole ("Unable to resolve host name")

    Else

        'Copy retrieved data to udtHostent structure
        RtlMoveMemory        udtHostent,        lngPtrToHOSTENT,
LenB(udtHostent)

        'Now udtHostent.hAddrList member contains
        'an array of IP addresses

        'Get a pointer to the first address
        RtlMoveMemory lngPtrToIP, udtHostent.hAddrList, 4

        Dim intCount As Integer
        intCount = 1

        Do Until lngPtrToIP = 0

            'Prepare the array to receive IP address values
            ReDim arrIpAddress(1 To udtHostent.hLength)

            'move IP address values to the array
            RtlMoveMemory       arrIpAddress(1),       lngPtrToIP,
udtHostent.hLength

            'build string with IP address
            For i = 1 To udtHostent.hLength
                strIpAddress  =  strIpAddress  &  arrIpAddress(i)  &
"."
            Next

            'remove the last dot symbol
            strIpAddress = Left$(strIpAddress, Len(strIpAddress) -
1)

            'Add IP address to the listbox
            arrIp(intCount) = strIpAddress
            intCount = intCount + 1

            'Clear the buffer
            strIpAddress = ""

            'Get pointer to the next address
            udtHostent.hAddrList    =    udtHostent.hAddrList    +
LenB(udtHostent.hAddrList)
            RtlMoveMemory lngPtrToIP, udtHostent.hAddrList, 4

        Loop

        intValidIp = intCount - 1
```

```
          intListenIP = intCount - 1
          intBridgeIP = 1

          'set ping ip functions initial values
          txtPingIP = arrIp(intListenIP)
          txtPingPort.Text = "552"

     End If

     showIPAddresses

End Function

Private Function showIPAddresses()

     'print the valid ip addresses to the form
     txtIPaddress.Caption = "Available IP Addresses:" + vbCrLf

     Dim intCount As Integer
     intCount = 1

     For intCount = 1 To 6

         If Not IsNull(arrIp(intCount)) Then

             'show each valid ip address in the address pool
             txtIPaddress.Caption     =     txtIPaddress.Caption     +
arrIp(intCount)
             If intCount = intListenIP Then txtIPaddress.Caption =
txtIPaddress.Caption + " (listen)"
             If intCount = intListenIP Then txtIPaddress.Caption =
txtIPaddress.Caption + " (bridge)"
             txtIPaddress.Caption = txtIPaddress.Caption + vbCrLf

         End If

     Next intCount

End Function

Private    Function    bindOneSocket(lngTargetSocket    As    Long,
strTargetIP As String, lngTargetPort As Long)

     Dim lngRetValue As Long

     'Begin binding the socket to the correct IP
     lngRetValue      =      vbBind(lngTargetSocket,      strTargetIP,
lngTargetPort)

     If lngRetValue = SOCKET_ERROR Then

         'If an error occurs, output the error to the console
         updateConsole ("Socket error: " & _
                         GetErrorDescription(Err.LastDllError))

     Else
```

```
        'If no errors occurs, display a confirmation
        'updateConsole  ("Socket  "  +  CStr(lngTargetSocket)  +  "
bound to " & _
                        strTargetIP + ":" + CStr(lngTargetPort))

    End If

End Function

Private Function listenOneSocket(lngTargetSocket As Long)

    Dim lngRetValue As Long

    'Begin listening on the assigned port
    lngRetValue = vbListen(lngTargetSocket)

    'Check a value returned by the vbListen function
    If lngRetValue = SOCKET_ERROR Then

        'An error occurs - display the error message
        updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    ElseIf lngRetValue = 0 Then

        'If no errors occurs, display a confirmation
        'updateConsole  ("Socket  " + CStr(lngTargetSocket)  +  "  now
listening")

    End If

End Function

Private Function startTalkSocket()

    'This is the handle of the socket to be created
    Dim lngSocket As Long

    Dim lngAddressFamily As Long
    Dim lngSocketType As Long
    Dim lngProtocol As Long

    lngAddressFamily = 2      '2 = AF_INET: inter-network address
family
    lngSocketType = 1       '1 = SOCK_STREAM: socket streaming
    lngProtocol = 6            '6 = IPPROTO_TCP: transfer control
protocol

    'Create the talk socket
    lngSocket        =        mdWinSock.vbSocket(lngAddressFamily,
lngSocketType, lngProtocol)
    lngTalkSocketHandle = lngSocket

    If lngSocket = INVALID_SOCKET Then
```

```
        'If the function has returned the INVALID_SOCKET
        'value the socket was not created.
        updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    Else

        'updateConsole ("Talk socket opened: " & _
                    CStr(lngTalkSocketHandle))
        txtTalkSocket.Caption    =    "Talk    socket:    "    +
CStr(lngTalkSocketHandle)

    End If

End Function

Private   Function   connectTalkSocket(lngTargetSocket   As   Long,
strTargetIP As String, intTargetPort As Long)

    Dim lngRetValue As Long

    'Call  the  vbConnect  function  in  order  to  establish  the
connection
    lngRetValue      =      mdWinSock.vbConnect(lngTargetSocket,
strTargetIP, intTargetPort)

    If lngRetValue = SOCKET_ERROR Then

        'If the function has returned the INVALID_SOCKET
        'value the socket was not connected.
        updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    Else

        'The connection was established successfully.
        'updateConsole   ("Connection   request:   socket   "   +
CStr(lngTargetSocket) + " to " & _
                    strTargetIP + ":" + CStr(intTargetPort))
        txtTalkSocket = "Talk socket: " + CStr(lngTargetSocket) +
" (CONNECTED)"

    End If


End Function

Private Function sendData(strMessage As String)

    'Call the vbSend function in order to send data
    If vbSend(lngTalkSocketHandle, strMessage) = SOCKET_ERROR Then

        'If the vbSend function has returned a value of
        'SOCKET_ERROR, console displays the socket error
        updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))
    Else

        'If execution is successful, clear the textbox
        updateConsole    ("Message    sent    by    socket    "    +
CStr(lngTalkSocketHandle))
        'updateConsole ("Sent by socket " & _
                        CStr(lngTalkSocketHandle)    &    ":"    &
strMessage)

        'close the talk socket and create a new one
        Call closesocket(lngTalkSocketHandle)
        'updateConsole     ("Refreshing     socket     "     +
CStr(lngTalkSocketHandle))
        lngTalkSocketHandle = 0
        txtTalkSocket.Caption = "Talk socket: none"

        startTalkSocket

    End If

End Function

Private Sub tmrSocketCheck_Timer()

    Dim udtRead_fd As fd_set
    Dim udtWrite_fd As fd_set
    Dim udtError_fd As fd_set
    Dim lngSocketCount As Long
    Dim incomingConnection As Boolean
    Dim lngRetValue As Long
    Dim intCount As Integer

    'Interval Event 1: Listen Socket Checker

    'check to see if any clients are attempting to make a
    'connection attempt if a socket is defined

    If lngListenSocketHandle > 0 Then

        'checks listen socket to see if data is readable
        udtRead_fd.fd_count = 1
        udtRead_fd.fd_array(1) = lngListenSocketHandle

        lngSocketCount   =   vbselect(0&,   udtRead_fd,   udtWrite_fd,
udtError_fd, 0&)

        incomingConnection = CBool(lngSocketCount)

        If incomingConnection = True Then

            'If  an  incoming  connection  is  detected,  output  to
console
            updateConsole   ("Incoming   connection   on   "   +
CStr(lngListenSocketHandle))
```

```
            'Call the vbAccept function in order to accept the
            'connection request and create a new socket
            lngRetValue = vbAccept(lngListenSocketHandle)

            If lngRetValue = INVALID_SOCKET Then

                'An error has occurred - show the error message
                updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

            Else

                createSubsocket (lngRetValue)

            End If

        End If

    End If

    'Interval Event 2: Bridge Socket Checker

    'check to see if any clients are attempting to make a
    'connection attempt if a socket is defined

    If lngBridgeSocketHandle > 0 Then

        'checks listen socket to see if data is readable
        udtRead_fd.fd_count = 1
        udtRead_fd.fd_array(1) = lngBridgeSocketHandle

        lngSocketCount  =  vbselect(0&,  udtRead_fd,  udtWrite_fd,
udtError_fd, 0&)

        incomingConnection = CBool(lngSocketCount)

        If incomingConnection = True Then

            'If  an  incoming  connection  is  detected,  output  to
console
            updateConsole   ("Incoming   connection   on   "   +
CStr(lngBridgeSocketHandle))

            'Call the vbAccept function in order to accept the
            'connection request and create a new socket
            lngRetValue = vbAccept(lngBridgeSocketHandle)

            If lngRetValue = INVALID_SOCKET Then

                'An error has occurred - show the error message
                updateConsole ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

            Else

                createSubsocket (lngRetValue)
            End If

        End If

    End If

    'Interval Event 3: Subsocket Checker

    'check  to  see  if  subsockets  have  any  information  pending
retrieval

    For intCount = 1 To intMaxSockets

        If arrSubsocketHandles(intCount) > 0 Then

            'checks listen socket to see if data is readable
            udtRead_fd.fd_count = 1
            udtRead_fd.fd_array(1) = arrSubsocketHandles(intCount)

            lngSocketCount = vbselect(0&, udtRead_fd, udtWrite_fd,
udtError_fd, 0&)

            incomingConnection = CBool(lngSocketCount)

            If incomingConnection = True Then

                'If a subsocket contains data, output to console
                emptySubsocket (arrSubsocketHandles(intCount))

            End If

        End If

    Next intCount


End Sub

Private Function createSubsocket(lngSocketHandle As Long)

    'creates a subsocket when a socket request is accepted
    'search for an empty subsocket slot
    Dim intCount As Integer
    Dim placedSocket As Boolean

    placedSocket = False

    For intCount = 1 To intMaxSockets

        If  arrSubsocketHandles(intCount)  =  0  And  placedSocket  =
False Then

            arrSubsocketHandles(intCount) = lngSocketHandle
            placedSocket = True
            'connection accepted
            updateConsole ("Subsocket detected: socket " & _
                        CStr(lngSocketHandle))

        End If

    Next intCount

    refreshSubsockets

End Function

Private Sub refreshSubsockets()

    Dim intCount As Integer

    'Display subsockets in the server window
    txtSubsockets.Caption = ""
    For intCount = 1 To intMaxSockets

        If arrSubsocketHandles(intCount) > 0 Then
            txtSubsockets.Caption = txtSubsockets.Caption & _
                    CStr(arrSubsocketHandles(intCount)) + vbCrLf
        End If

    Next intCount

End Sub

Private Sub emptySubsocket(lngSocketHandle As Long)

    Dim strMsg As String
    Dim lngBytesReceived As Long
    Dim intCount As Integer

    'Call the vbRecv function to read data
    lngBytesReceived = vbRecv(lngSocketHandle, strMsg)

    If lngBytesReceived > 0 Then

        'If we have received some data, put it into the console
and
        'close the subsocket
        updateConsole    ("Message    received    from     "     +
CStr(lngSocketHandle))
        'updateConsole ("From " + CStr(lngSocketHandle) + ": " +
strMsg)
        handleAction (strMsg)

        For intCount = 1 To 16

        If arrSubsocketHandles(intCount) = lngSocketHandle Then

            Call closesocket(arrSubsocketHandles(intCount))
            updateConsole     ("Subsocket     closed:     "      +
CStr(arrSubsocketHandles(intCount)))
            arrSubsocketHandles(intCount) = 0
            refreshSubsockets

        End If

    Next intCount

    ElseIf lngBytesReceived = SOCKET_ERROR Then

        'An error has occurred - show the error message
        updateConsole ("Socket error: " & _
            GetErrorDescription(Err.LastDllError))

    End If

End Sub


Private Sub handleAction(strMsg As String)

    If InStr(strMsg, "action::") > 0 Then

        Dim fields() As String
        fields() = Split(strMsg, "::")

        Dim actionTaken As String
        Dim retValue As Integer

        actionTaken = fields(2)

        Select Case actionTaken

            Case "createAccount"
                retValue    =   createAccount(fields(1),   fields(3),
fields(4), fields(5))
            Case "loginAccount"
                retValue    =   loginAccount(fields(1),   fields(3),
fields(4))
            Case "updateAccount"
                retValue    =   updateAccount(fields(1),   fields(3),
fields(4), fields(5))

        End Select

    End If

End Sub

Private  Function  createAccount(IPAddress  As  String,  matricNumber
As String, StudentName As String, password As String)

    Dim dataFilename As String
    Dim retValue As Integer

    findUser (matricNumber)
    'accountExists = False
    If accountExists Then

        updateConsole ("Account creation failed: this account
already exists!")
        retValue = informClient(IPAddress, "info::Error creating
account: Account already exists!")

    Else
        'create a new user entry

        totalStudentRecords = totalStudentRecords + 1

        StudentData.StudentName = StudentName
        StudentData.StudentMatric = matricNumber
        StudentData.StudentIC = password
        StudentData.AccountActive = False
        StudentData.FirstRecordIndex = 0
        StudentData.StudentIndex = totalStudentRecords

        Put StudentFile, totalStudentRecords, StudentData

        updateConsole ("New account created: " & matricNumber)
        retValue = informClient(IPAddress, "info::Account creation
successful!")

    End If

End Function

Private Function findUser(searchMatric As String)

    Dim currentIndex As Long
    currentIndex = 0
    accountExists = False

    If Not LOF(StudentFile) = 0 Then

        lstUsers.ListItems.Clear
        While ((currentIndex < totalStudentRecords) And (Not
accountExists))
            currentIndex = currentIndex + 1
            Get StudentFile, currentIndex, StudentData

            If (StrComp(StudentData.StudentMatric, searchMatric) =
0) Then
                accountExists = True
            End If
        Wend

    End If

End Function

Private Function showAllUsers()

    Dim currentIndex As Long
    currentIndex = 0

    'refresh user file
    If Not LOF(StudentFile) = 0 Then

        lstUsers.ListItems.Clear
        While currentIndex < totalStudentRecords
            currentIndex = currentIndex + 1
            Get StudentFile, currentIndex, AllStudentData

            lstUsers.ListItems.Add                ,                ,
Trim$(AllStudentData.StudentName)
            lstUsers.ListItems(currentIndex).SubItems(1)           =
CStr(AllStudentData.StudentMatric)
            lstUsers.ListItems(currentIndex).SubItems(2)           =
CStr(AllStudentData.AccountActive)
            lstUsers.ListItems(currentIndex).SubItems(3)           =
CStr(AllStudentData.FirstRecordIndex)
        Wend

    End If

    'refresh account file
    currentIndex = 0
    If Not LOF(AccountFile) = 0 Then

        lstAccount.ListItems.Clear
        While currentIndex < totalAccountRecords
            currentIndex = currentIndex + 1
            Get AccountFile, currentIndex, AllAccountData

            lstAccount.ListItems.Add                ,                ,
CStr(AllAccountData.prevRecord)
            lstAccount.ListItems(currentIndex).SubItems(1)         =
CStr(AllAccountData.itemDate)
            lstAccount.ListItems(currentIndex).SubItems(2)         =
CStr(AllAccountData.itemStatus)
            lstAccount.ListItems(currentIndex).SubItems(3)         =
Trim$(AllAccountData.itemName)
            lstAccount.ListItems(currentIndex).SubItems(4)         =
CStr(AllAccountData.itemQuantity)
            lstAccount.ListItems(currentIndex).SubItems(5)         =
CStr(AllAccountData.nextRecord)
        Wend

    End If

End Function

Private Function updateAccount(IPAddress As String, matricNumber
As String, accountInfo As String, numberOfItems As String)

    Dim retValue As Integer

    Dim fields() As String
    Dim accountfields() As String
    Dim accountdatafields() As String
    Dim currentItem As Long

    Dim prevRecord As Long
    Dim currentRecord As Long
    Dim nextRecord As Long

    Dim firstRecord As Boolean

    Dim thisDate As String
    Dim thisStatus As String
    Dim thisName As String
    Dim thisQuantity As String

    Dim strLine As String
    Dim strCompiled As String
    Dim currentIndex As Long
    Dim allItemsLoaded As Boolean

    'update account file using the list submitted by the client
    If CLng(numberOfItems) > 0 Then
        accountfields() = Split(accountInfo, "++")
    End If

    findUser (matricNumber)

    'check if this student has any records
    If StudentData.FirstRecordIndex = 0 Then
        firstRecord = True
    Else
        currentRecord = 0
        nextRecord = StudentData.FirstRecordIndex
        firstRecord = False
    End If

    'loop through all items
    While currentItem < CLng(numberOfItems)

        accountdatafields()   =   Split(accountfields(currentItem),
",")

        'if this student has no existing records, create one
        If firstRecord Then

            totalAccountRecords = totalAccountRecords + 1
            StudentData.FirstRecordIndex = totalAccountRecords

            'overwrite to point to the new record
            Put StudentFile, StudentData.StudentIndex, StudentData

            AccountData.isFirst = True
            AccountData.isLast = True
            AccountData.prevRecord = 0
            AccountData.nextRecord = 0

            'collect the data from the strings
            AccountData.itemDate = accountdatafields(0)
            If StrComp(accountdatafields(1), "Borrowed") = 0 Then
                AccountData.itemStatus = False
            Else
                AccountData.itemStatus = True
            End If
            AccountData.itemName = accountdatafields(2)
            AccountData.itemQuantity = accountdatafields(3)

            Put AccountFile, totalAccountRecords, AccountData

            firstRecord = False
            prevRecord = 0
            currentRecord = totalAccountRecords
            nextRecord = 0

        'if the student already has existing records,
        Else

            'move on to the next record
            prevRecord = currentRecord
            currentRecord = nextRecord

            'end of existing records, time to add new ones.
            If currentRecord = 0 Then

                'link previous record to this one
                If Not prevRecord = 0 Then

                    Get AccountFile, prevRecord, AccountData
                    AccountData.isLast = False
                    AccountData.nextRecord = totalAccountRecords +
1

                    Put AccountFile, prevRecord, AccountData

                End If

                totalAccountRecords = totalAccountRecords + 1
                AccountData.isFirst = False
                AccountData.isLast = True
                AccountData.prevRecord = prevRecord
                AccountData.nextRecord = 0

                'collect the data from the strings
                AccountData.itemDate = accountdatafields(0)
                If  StrComp(accountdatafields(1),  "Borrowed")  =  0
Then
                    AccountData.itemStatus = False
                Else
                    AccountData.itemStatus = True
                End If
                AccountData.itemName = accountdatafields(2)
                AccountData.itemQuantity = accountdatafields(3)

                Put AccountFile, totalAccountRecords, AccountData
                firstRecord = False
                currentRecord = totalAccountRecords
                nextRecord = 0

            'the next record still exists
            Else

                Get AccountFile, currentRecord, AccountData

                'change the contents
                AccountData.itemDate = accountdatafields(0)
                If  StrComp(accountdatafields(1),  "Borrowed")  =  0
Then
                    AccountData.itemStatus = False
                Else
                    AccountData.itemStatus = True
                End If
                AccountData.itemName = accountdatafields(2)
                AccountData.itemQuantity = accountdatafields(3)

                Put AccountFile, currentRecord, AccountData

                'find the next record
                nextRecord = AccountData.nextRecord

            End If

        End If

        currentItem = currentItem + 1
    Wend

    retValue = informClient(IPAddress, "updateok::")

    'Move all data back to the client
    If StudentData.FirstRecordIndex > 0 Then

        'loop through the record indexes
        currentIndex = StudentData.FirstRecordIndex
        allItemsLoaded = False
        strCompiled = ""

        While Not allItemsLoaded

            Get AccountFile, currentIndex, AccountData

            strLine = ""
            strLine = strLine & CStr(currentIndex) & ",,"
            strLine = strLine & CStr(AccountData.itemDate) & ",,"
            If AccountData.itemStatus Then
                strLine = strLine & "True" & ",,"
            Else
                strLine = strLine & "False" & ",,"
            End If
            strLine = strLine & CStr(AccountData.itemName) & ",,"
            strLine = strLine & CStr(AccountData.itemQuantity)
            strCompiled = strCompiled & strLine & "++"

            currentIndex = AccountData.nextRecord

            'until all the items have been found
            If AccountData.isLast Then
                allItemsLoaded = True
            End If

        Wend

        retValue    =    informClient(IPAddress,    "write::"    &
strCompiled)

    End If

    updateConsole ("Account updated: " & matricNumber)
    retValue   =   informClient(IPAddress,   "info::Account   update
successful!")

End Function

Private Function fileExists(ByVal filename As String)

    Dim length As Long

    On Error GoTo FileDoesntExist

        length = FileLen(filename)
        fileExists = True

    Exit Function

FileDoesntExist:
        fileExists = False

End Function

Private Function loginAccount(IPAddress As String, matricNumber As
String, password As String)

    Dim retValue As Integer
    Dim strCompiled As String
    Dim strLine As String
    Dim endOfAccount As Boolean
    Dim nextIndex As Long
    Dim currentIndex As Integer
    Dim allItemsLoaded As Boolean

    findUser (matricNumber)
    If Not accountExists Then

        updateConsole  ("Account  login  failed:  this  account  does
not exist!")
        retValue    =    informClient(IPAddress,    "info::Invalid
account.")
```

```
    Else

        If  (StrComp(Trim$(StudentData.StudentIC),  password)  =  0)
Then
            'if password is correct
            updateConsole ("Login successful: " & matricNumber)
            retValue  =  informClient(IPAddress,  "info::Account
login successful!")

            'if login is succesful, dump the entire file to the
client
            retValue = informClient(IPAddress, "clear::")

            'move user data to client
            retValue  =  informClient(IPAddress,  "setname::"  &
StudentData.StudentName)
            retValue  =  informClient(IPAddress,  "setmatric::"  &
StudentData.StudentMatric)
            If StudentData.AccountActive Then
                retValue        =        informClient(IPAddress,
"setactive::true")
            Else
                retValue        =        informClient(IPAddress,
"setactive::false")
            End If

            'move account data to client
            If StudentData.FirstRecordIndex > 0 Then

                'loop through the record indexes
                currentIndex = StudentData.FirstRecordIndex
                allItemsLoaded = False
                strCompiled = ""

                While Not allItemsLoaded

                    Get AccountFile, currentIndex, AccountData

                    strLine = ""
                    strLine = strLine & CStr(currentIndex) & ",,"
                    strLine = strLine & CStr(AccountData.itemDate)
& ",,"
                    If AccountData.itemStatus Then
                        strLine = strLine & "True" & ",,"
                    Else
                        strLine = strLine & "False" & ",,"
                    End If
                    strLine = strLine & CStr(AccountData.itemName)
& ",,"
                    strLine          =          strLine          &
CStr(AccountData.itemQuantity)
                    strCompiled = strCompiled & strLine & "++"

                    currentIndex = AccountData.nextRecord
                    'until all the items have been found
                    If AccountData.isLast Then
                        allItemsLoaded = True
                    End If

                Wend

                retValue  =  informClient(IPAddress,  "write::"  &
strCompiled)

            End If

            'inform the client that the file has been transferred
            retValue = informClient(IPAddress, "loginok::")

        Else

            'if password is incorrect
            updateConsole ("Login failed: " & matricNumber)
            retValue   =   informClient(IPAddress,   "info::Account
login failed!")

        End If
    End If

End Function
```

Appendix B: Client Software Source Codes (Main Menu)

```
'--------------------------------------------------
'Form     : fmClient
'Function : connects to the server and does all
'           client side data handling
'--------------------------------------------------
'
'By: Sim Yih Chun
'
'--------------------------------------------------

Option Explicit

'Prepare storage array for the last 10 events
Dim consoleEvents(10) As String

'Prepare array for system's ip addresses
'intValidIp stores the number of valid ip addresses found
'strActiveIp is a global variable that stores the active IP address
Dim arrIp(6) As String
Dim intValidIp As Integer
Dim intListenIP As Integer
Dim intMaxSockets As Integer

'Saves the socket handles as global variables
Dim lngListenSocketHandle As Long
Dim lngTalkSocketHandle As Long
Dim arrSubsocketHandles(16) As Long

'Return value handles in main procedures
Dim lngRetListenSocket As Long
Dim lngRetTalkSocket As Long

'Stores the port numbers for the listen IP
Dim lngListenPort As Long

'Stores the default connection values
Dim strServerIP As String
Dim intServerPort As Long

'user data file values
Dim userActive As String
Dim userName As String
Dim userMatric As String
Dim loggedIn As Boolean

'time out counter for msges
Dim msgTimer As Integer

'Define data type for the inventory database
Private Type AccountRecord
    itemIndex As Long
    itemDate As String * 10
    itemStatus As Boolean
    itemName As String * 50
    itemQuantity As Long
End Type

Dim AccountData As AccountRecord
Dim AllAccountData As AccountRecord
Dim AccountFile&
Dim AccountRecordLength&
Dim totalAccountRecords&

Private Sub btnBulletin_Click()

    'fmClient.Hide

End Sub

Private Sub btnExit_Click()

    stopAllSockets
    Unload Me

End Sub

Private Sub btnListenIP_Click()

    'Switch the active listen IP to the next available IP address

    stopAllSockets
    intListenIP = intListenIP + 1

    If intListenIP > intValidIp Then

        intListenIP = 1

    End If

    'recreate the listen/talk socket pair
    startListenSocket
    startTalkSocket


End Sub

Private Sub btnNewAccount_Click()

    Load frmCreateAcc
    frmCreateAcc.Show
    fmClient.Hide

End Sub


Private Sub btnLoginAccount_Click()

    Dim retValue As Integer
    'clears the user file
    Close AccountFile
    Open "ClientRecord.db" For Output As #2
    Close #2
    AccountFile = FreeFile
    AccountRecordLength = Len(AccountData)
    Open   "ClientRecord.db"   For   Random   As   AccountFile   Len   =
AccountRecordLength
    totalAccountRecords = LOF(AccountFile) \ AccountRecordLength

    'resolve login
    If loggedIn = False Then
        Load frmLoginAcc
        fmClient.Hide
        frmLoginAcc.Show
    Else
        loggedIn = False
        btnLoginAccount.Caption = "student login"
        retValue = logoutAccount()
    End If

End Sub

Private Sub btnInventory_Click()

    If loggedIn Then
        Close AccountFile
        Load frmInventory
        fmClient.Hide
        frmInventory.Show
    Else
        showErrorMsg ("Please login first!")
    End If

End Sub

Private Sub Form_Load()

    'initial values are set.
    txtStatus.Caption = "Disconnected"
    lngListenSocketHandle = 0
    intMaxSockets = 8
    loggedIn = False

    'initial button conditions are set
    btnInventory.Enabled = False
    btnBulletin.Enabled = False

    'set the date on the calendar
    lblDate.Caption = CStr(Day(Date))

    Dim thisMonth As Integer

    thisMonth = CInt(DatePart("m", Date))

    If thisMonth = 1 Then
```

```
        lblMonth.Caption = "January"
    ElseIf thisMonth = 2 Then
        lblMonth.Caption = "February"
    ElseIf thisMonth = 3 Then
        lblMonth.Caption = "March"
    ElseIf thisMonth = 4 Then
        lblMonth.Caption = "April"
    ElseIf thisMonth = 5 Then
        lblMonth.Caption = "May"
    ElseIf thisMonth = 6 Then
        lblMonth.Caption = "June"
    ElseIf thisMonth = 7 Then
        lblMonth.Caption = "July"
    ElseIf thisMonth = 8 Then
        lblMonth.Caption = "August"
    ElseIf thisMonth = 9 Then
        lblMonth.Caption = "September"
    ElseIf thisMonth = 10 Then
        lblMonth.Caption = "October"
    ElseIf thisMonth = 11 Then
        lblMonth.Caption = "November"
    ElseIf thisMonth = 12 Then
        lblMonth.Caption = "December"
    End If

    'set the bridge/listen port numbers
    lngListenPort = 552

    'value returned by the InitializeWinsock function
    Dim lngRetValue As Long

    'initialize the Winsock service
    lngRetValue = mdWinSock.InitializeWinsock(SOCKET_VERSION_22)

    If Not lngRetValue = 0 Then

        'if the Winsock service was not initialized
        'successfully, show the error
        showErrorMsg ("Unable to initialise WinSock API!")

    End If

    getIPAddresses

    'if the Winsock service was initialized
    'successfully, create listen/talk socket pair

    startListenSocket
    startTalkSocket
    loadDefaultFile

    'empty the random access file
    Open "ClientRecord.db" For Output As #2
    Close #2

    'prepare the random access file for the temporary account
    AccountFile = FreeFile
    AccountRecordLength = Len(AccountData)
    Open  "ClientRecord.db"  For  Random  As  AccountFile  Len  =
AccountRecordLength
    totalAccountRecords = LOF(AccountFile) \ AccountRecordLength

End Sub

Private Sub Form_Unload(Cancel As Integer)

    Call WSACleanup

End Sub

Private Sub menuExit_Click()

    Unload Me

End Sub

Private Function showErrorMsg(errorEvent As String)

    Dim intResponse As Integer

    intResponse = MsgBox(errorEvent, vbExclamation, "Error!")

End Function

Private Function startListenSocket()

    'Update status display and console
    txtStatus.Caption = "Starting..."

    'This is the handle of the socket to be created
    Dim lngSocket As Long

    Dim lngAddressFamily As Long
    Dim lngSocketType As Long
    Dim lngProtocol As Long

    lngAddressFamily  =  2      '2  =  AF_INET:  inter-network  address
family
    lngSocketType = 1       '1 = SOCK_STREAM: socket streaming
    lngProtocol  =  6              '6  =  IPPROTO_TCP:  transfer  control
protocol

    'Create the sockets: 1 bridge socket between servers and 1 listen
socket
    lngSocket  =  mdWinSock.vbSocket(lngAddressFamily,  lngSocketType,
lngProtocol)
    lngListenSocketHandle = lngSocket

    If lngSocket = INVALID_SOCKET Then

        'If the function has returned the INVALID_SOCKET
        'value the socket was not created. Call the
        'ShowErrorMsg subroutine to show the message box
        'with the error description.
        txtStatus.Caption = "Disconnected"
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    Else

        'If a new socket was created successfully, add the
        'listview's item for that socket
        txtStatus.Caption = "Client IP: (" + arrIp(intListenIP) + ":
Port " + CStr(lngListenPort) & _
                "- socket " + CStr(lngListenSocketHandle) + ")"
        lngRetListenSocket   =   bindOneSocket(lngListenSocketHandle,
arrIp(intListenIP), lngListenPort)
        lngRetListenSocket = listenOneSocket(lngListenSocketHandle)

    End If

End Function

Private Function stopAllSockets()

    'shuts down the listen server
    'close the listening socket handle
    Call closesocket(lngListenSocketHandle)
    lngListenSocketHandle = 0

    Call closesocket(lngTalkSocketHandle)
    lngTalkSocketHandle = 0

    Dim intCount As Integer

    For intCount = 1 To 16

        If arrSubsocketHandles(intCount) > 0 Then

            Call closesocket(arrSubsocketHandles(intCount))
            arrSubsocketHandles(intCount) = 0

        End If

    Next intCount

    txtStatus.Caption = "Disconnected"

End Function


Private   Sub   Form_QueryUnload(Cancel   As   Integer,   UnloadMode   As
Integer)

    If Not lngListenSocketHandle = 0 Then

        stopAllSockets
    End If

End Sub

Private Function getIPAddresses()

    'pointer to HOSTENT structure returned by
    'the gethostbyname function
    Dim lngPtrToHOSTENT As Long

    'structure which stores all the host info
    Dim udtHostent      As HOSTENT

    'pointer to the IP address' list
    Dim lngPtrToIP      As Long

    'byte array that contains elemets of an IP address
    Dim arrIpAddress()  As Byte

    'result IP address string to add into the ListBox
    Dim strIpAddress    As String

    'buffer string to receive the local system host name
    Dim strHostName As String * 256
    'value returned by the gethostname function
    Dim lngRetVal As Long
    Dim i As Integer

    'Get the local host name
    lngRetVal = gethostname(strHostName, 256)

    If lngRetVal = SOCKET_ERROR Then
        MsgBox "Can't resolve the local host address", vbExclamation
        Exit Function
    End If

    'Call the gethostbyname Winsock API function
    'to get pointer to the HOSTENT structure
    lngPtrToHOSTENT    =    gethostbyname(Left(strHostName,    InStr(1,
strHostName, Chr(0)) - 1))

    'Check the lngPtrToHOSTENT value
    If lngPtrToHOSTENT = 0 Then

        'If the gethostbyname function has returned 0
        'the function execution is failed.

        showErrorMsg ("Unable to resolve host name")

    Else

        'Copy retrieved data to udtHostent structure
        RtlMoveMemory udtHostent, lngPtrToHOSTENT, LenB(udtHostent)

        'Now udtHostent.hAddrList member contains
        'an array of IP addresses
        'Get a pointer to the first address
        RtlMoveMemory lngPtrToIP, udtHostent.hAddrList, 4

        Dim intCount As Integer
        intCount = 1

        Do Until lngPtrToIP = 0

            'Prepare the array to receive IP address values
            ReDim arrIpAddress(1 To udtHostent.hLength)

            'move IP address values to the array
            RtlMoveMemory        arrIpAddress(1),        lngPtrToIP,
udtHostent.hLength

            'build string with IP address
            For i = 1 To udtHostent.hLength
                strIpAddress = strIpAddress & arrIpAddress(i) & "."
            Next

            'remove the last dot symbol
            strIpAddress = Left$(strIpAddress, Len(strIpAddress) - 1)

            'Add IP address to the listbox
            arrIp(intCount) = strIpAddress
            intCount = intCount + 1

            'Clear the buffer
            strIpAddress = ""

            'Get pointer to the next address
            udtHostent.hAddrList     =     udtHostent.hAddrList     +
LenB(udtHostent.hAddrList)
            RtlMoveMemory lngPtrToIP, udtHostent.hAddrList, 4

        Loop

        intValidIp = intCount - 1
        intListenIP = intCount - 1

    End If

End Function


Private Function bindOneSocket(lngTargetSocket As Long, strTargetIP
As String, lngTargetPort As Long)

    Dim lngRetValue As Long

    'Begin binding the socket to the correct IP
    lngRetValue = vbBind(lngTargetSocket, strTargetIP, lngTargetPort)

    If lngRetValue = SOCKET_ERROR Then
```

```
        'If an error occurs, output the error to the console
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    End If

End Function

Private Function listenOneSocket(lngTargetSocket As Long)

    Dim lngRetValue As Long

    'Begin listening on the assigned port
    lngRetValue = vbListen(lngTargetSocket)

    'Check a value returned by the vbListen function
    If lngRetValue = SOCKET_ERROR Then

        'An error occurs - display the error message
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    End If

End Function

Private Function startTalkSocket()

    'This is the handle of the socket to be created
    Dim lngSocket As Long

    Dim lngAddressFamily As Long
    Dim lngSocketType As Long
    Dim lngProtocol As Long

    lngAddressFamily = 2      '2 = AF_INET: inter-network address
family
    lngSocketType = 1       '1 = SOCK_STREAM: socket streaming
    lngProtocol = 6            '6 = IPPROTO_TCP: transfer control
protocol

    'Create the talk socket
    lngSocket = mdWinSock.vbSocket(lngAddressFamily, lngSocketType,
lngProtocol)
    lngTalkSocketHandle = lngSocket

    If lngSocket = INVALID_SOCKET Then

        'If the function has returned the INVALID_SOCKET
        'value the socket was not created.
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    End If

End Function
```

```
Private Function connectTalkSocket()

    Dim lngRetValue As Long

    'Call the vbConnect function in order to establish the connection
    lngRetValue      =      mdWinSock.vbConnect(lngTalkSocketHandle,
strServerIP, intServerPort)

    If lngRetValue = SOCKET_ERROR Then

        'If the function has returned the INVALID_SOCKET
        'value the socket was not connected.
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    End If


End Function

Private Function sendData(strMessage As String)

    connectTalkSocket

    'Call the vbSend function in order to send data
    If vbSend(lngTalkSocketHandle, strMessage) = SOCKET_ERROR Then

        'If the vbSend function has returned a value of
        'SOCKET_ERROR, console displays the socket error
        showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

    Else

        'close the talk socket and create a new one
        Call closesocket(lngTalkSocketHandle)
        lngTalkSocketHandle = 0

        startTalkSocket

    End If

End Function
Private Sub tmrSocketCheck_Timer()

    Dim udtRead_fd As fd_set
    Dim udtWrite_fd As fd_set
    Dim udtError_fd As fd_set
    Dim lngSocketCount As Long
    Dim incomingConnection As Boolean
    Dim lngRetValue As Long
    Dim intCount As Integer

    'Interval Event 1: Listen Socket Checker
    'check to see if any clients are attempting to make a
    'connection attempt if a socket is defined

    If lngListenSocketHandle > 0 Then

        'checks listen socket to see if data is readable
        udtRead_fd.fd_count = 1
        udtRead_fd.fd_array(1) = lngListenSocketHandle

        lngSocketCount  =  vbselect(0&,  udtRead_fd,  udtWrite_fd,
udtError_fd, 0&)

        incomingConnection = CBool(lngSocketCount)

        If incomingConnection = True Then

            'Call the vbAccept function in order to accept the
            'connection request and create a new socket
            lngRetValue = vbAccept(lngListenSocketHandle)

            If lngRetValue = INVALID_SOCKET Then

                'An error has occurred - show the error message
                showErrorMsg ("Socket error: " & _
                        GetErrorDescription(Err.LastDllError))

            Else

                createSubsocket (lngRetValue)

            End If

        End If

    End If

    'Interval Event 2: Subsocket Checker

    'check  to  see  if  subsockets  have  any  information  pending
retrieval

    For intCount = 1 To intMaxSockets

        If arrSubsocketHandles(intCount) > 0 Then

            'checks listen socket to see if data is readable
            udtRead_fd.fd_count = 1
            udtRead_fd.fd_array(1) = arrSubsocketHandles(intCount)

            lngSocketCount  =  vbselect(0&,  udtRead_fd,  udtWrite_fd,
udtError_fd, 0&)

            incomingConnection = CBool(lngSocketCount)

            If incomingConnection = True Then
```

```
                'If a subsocket contains data, output to console
                emptySubsocket (arrSubsocketHandles(intCount))

            End If

        End If

    Next intCount

    'Internal Event 3: Perform Pending Actions

    If Not pendingAction = "" Then

        Dim message As String

        message = "action"
        message = message & "::" & arrIp(intListenIP)
        message = message & "::" & pendingAction
        message = message & "::" & passVariable1
        message = message & "::" & passVariable2
        message = message & "::" & passVariable3
        message = message & "::" & passVariable4

        sendData (message)

        pendingAction = ""

    End If

    'clear message timer
    If msgTimer > 0 Then
        msgTimer = msgTimer - 1
        If msgTimer = 0 Then
            If loggedIn Then
                lblUser = "Logged in as: " & userName
            Else
                lblUser = "You are currently not logged in!"
            End If
        End If
    End If

End Sub

Private Function createSubsocket(lngSocketHandle As Long)

    'creates a subsocket when a socket request is accepted
    'search for an empty subsocket slot
    Dim intCount As Integer
    Dim placedSocket As Boolean

    placedSocket = False

    For intCount = 1 To intMaxSockets

        If arrSubsocketHandles(intCount) = 0 And placedSocket = False
Then
```

```
            arrSubsocketHandles(intCount) = lngSocketHandle
            placedSocket = True

        End If

    Next intCount

End Function


Private Sub emptySubsocket(lngSocketHandle As Long)

    Dim strMsg As String
    Dim lngBytesReceived As Long
    Dim intCount As Integer
    Dim fields() As String
    Dim accountfields() As String
    Dim datafields() As String
    Dim strFileLine As String
    Dim retValue As Integer

    'Call the vbRecv function to read data
    lngBytesReceived = vbRecv(lngSocketHandle, strMsg)

    If lngBytesReceived > 0 Then

        'If we have received some data, put it into the console and
        'close the subsocket
        If InStr(strMsg, "info::") Then
            fields() = Split(strMsg, "::")
            lblUser.Caption = fields(1)
            'retValue = MsgBox(fields(1), vbOKOnly + vbInformation,
"E&E Scatternet Client")
            msgTimer = 25
        End If

        'If this is a clear command, clear the user file
        If InStr(strMsg, "clear::") Then
            Open "data/user" For Output As #2
            Close #2
        End If

        'If this is a setname command, then set the user name
        If InStr(strMsg, "setname::") Then
            fields() = Split(strMsg, "::")
            userName = fields(1)
        End If

        'If this is a setmatric command, then set the user matric
number
        If InStr(strMsg, "setmatric::") Then
            fields() = Split(strMsg, "::")
            userMatric = fields(1)
        End If
        'If this is a setactive command, then change the status of
the account
        If InStr(strMsg, "setactive::") Then
            fields() = Split(strMsg, "::")
            userActive = fields(1)
        End If

        'If this is a write command, write to user file
        If InStr(strMsg, "write::") Then
            fields() = Split(strMsg, "::")
            strMsg = fields(1)
            datafields() = Split(strMsg, "++")
            Dim i As Integer
            For i = 0 To UBound(datafields) - 1
                totalAccountRecords = totalAccountRecords + 1

                accountfields() = Split(datafields(i), ",,")

                AccountData.itemIndex = CLng(accountfields(0))
                AccountData.itemDate = accountfields(1)
                If StrComp(accountfields(2), "True") = 0 Then
                    AccountData.itemStatus = True
                Else
                    AccountData.itemStatus = False
                End If
                AccountData.itemName = accountfields(3)
                AccountData.itemQuantity = CLng(accountfields(4))

                Put AccountFile, totalAccountRecords, AccountData
            Next
        End If

        If InStr(strMsg, "loginok::") Then

            lblUser = "Logged in as: " & userName
            loggedIn = True

            currentUserName = userName
            currentUserMatric = userMatric
            currentUserActive = userActive

            btnLoginAccount.Caption = "student logout"
            btnNewAccount.Enabled = False
            btnInventory.Enabled = True
            btnBulletin.Enabled = True

        End If

        If InStr(strMsg, "updateok::") Then

            'prepare the random access file for the temporary account
            AccountFile = FreeFile
            AccountRecordLength = Len(AccountData)
            Open "ClientRecord.db" For Random As AccountFile Len =
AccountRecordLength
            totalAccountRecords      =       LOF(AccountFile)       \
AccountRecordLength

        End If

        For intCount = 1 To 16

        If arrSubsocketHandles(intCount) = lngSocketHandle Then

            Call closesocket(arrSubsocketHandles(intCount))
            arrSubsocketHandles(intCount) = 0

        End If

    Next intCount

    ElseIf lngBytesReceived = SOCKET_ERROR Then

        'An error has occurred - show the error message
        showErrorMsg ("Socket error: " & _
            GetErrorDescription(Err.LastDllError))

    End If

End Sub

Private Sub loadDefaultFile()

    'load default values from the config file
    Dim strConfigFilename As String
    Dim intConfigFile As Integer
    Dim strFileLine As String
    Dim strOption As Variant

    strConfigFilename = "default.snt"
    intConfigFile = FreeFile()

    'reads the config file line by line until EOF
    Open strConfigFilename For Input As intConfigFile
    Do Until EOF(intConfigFile)

        Line Input #1, strFileLine

        'only extract variables if = is found
        If strFileLine Like "*=*" Then

            strOption = Split(strFileLine, "=", 2)
            Select Case strOption(0)

            Case "[ServerAddress]"
                strServerIP = strOption(1)
                lblServerInfo.Caption = "Server IP:" + strServerIP
            Case "[ServerPort]"
                intServerPort = CInt(strOption(1))

            End Select
        End If

    Loop
    Close intConfigFile

End Sub

Private Function logoutAccount()

    lblUser = "Logging out"

    'disables the buttons
    btnNewAccount.Enabled = True
    btnInventory.Enabled = False
    btnBulletin.Enabled = False

    'clears the user file
    Close AccountFile
    Open "ClientRecord.db" For Output As #2
    Close #2
    AccountFile = FreeFile
    AccountRecordLength = Len(AccountData)
    Open   "ClientRecord.db"   For   Random   As   AccountFile   Len   =
AccountRecordLength
    totalAccountRecords = LOF(AccountFile) \ AccountRecordLength

    lblUser = "You are currently not logged in!"

End Function

Private Function loadVerifyCode()

    verifyPasswordFile = FreeFile()
    Open "data\verifycode.pwd" For Input As verifyPasswordFile
    Line Input #verifyPasswordFile, veriRawCode
    Close verifyPasswordFile

    Dim codeOnly As String
    Dim shuffledCode As String
    Dim hashCode As String
    Dim originalScore As String

    codeOnly = Mid(veriRawCode, 7, 6) & Mid(veriRawCode, 14, 6)
    shuffledCode = reshuffleCode(codeOnly, -1)
    hashCode = hashName(Left(shuffledCode, 10))

    originalScore = obfuscateString(Left(shuffledCode, 4), 36, 10, 7)
    veriTempCode = CStr(CLng(originalScore))

    'Check the validity of the code
    If StrComp(hashCode, Right(veriRawCode, 2)) = 0 Then
        verifyCode = veriTempCode
        verifyCodeValid = True
    Else
        verifyCodeValid = False
        Unload Me
    End If

End Function

Private Function generateCode() As String

    Dim scoreString As String
    Dim medalStringOne As String
    Dim medalStringTwo As String
    Dim rawCode As String
    Dim footerCode As String
    Dim finalCode As String

    scoreString  =  obfuscateString(CStr(CLng(veriTempCode)),  10,  36,
4)
    medalStringOne = obfuscateString("000000", 8, 36, 3)
    medalStringTwo = obfuscateString("000000", 8, 36, 3)

    rawCode = scoreString & medalStringOne & medalStringTwo
    footerCode = hashName(rawCode)

    finalCode = rawCode & footerCode
    finalCode = reshuffleCode(finalCode, 1)

    generateCode = Left(finalCode, 6) & "-" & Right(finalCode, 6)

End Function

Private  Function  reshuffleCode(inputString  As  String,  direction  As
Long) As String

    'Verification Code Hashing: general shuffle function
    Dim currentCount As Long
    Dim activeString As Long
    Dim firstChar As String
    Dim secondChar As String
    Dim finalString As String

    currentCount = 1
    activeString = Len(inputString) - 2
    firstChar = Mid(inputString, activeString + 1, 1)
    secondChar = Mid(inputString, activeString + 2, 1)
    finalString = ""

    While currentCount <= activeString
        firstChar = twistLetters(firstChar, secondChar, 1)
        finalString  =  finalString  &  twistLetters(Mid(inputString,
currentCount, 1), firstChar, direction)
        currentCount = currentCount + 1
    Wend
    reshuffleCode = finalString & Right(inputString, 2)

End Function

Private  Function  twistLetters(charOne  As  String,  charTwo  As  String,
direction As Long) As String
```

```
    Dim integerA As Long
    Dim integerB As Long
    Dim integerR As Long

    integerA = getCharIndex(charOne)
    integerB = getCharIndex(charTwo)
    integerR = integerA + integerB * direction
    If integerR < 0 Then
        integerR = 36 + integerR
    End If
    If integerR > 35 Then
        integerR = integerR - 36
    End If

    twistLetters = getChar(integerR)

End Function

Private Function hashName(inputString As String) As String

    Dim generatedNumber As Long
    Dim fullString As String

    fullString = modifyName(veriTempName) & inputString

    generatedNumber = generateHashNo(fullString)
    hashName = obfuscateString(CStr(generatedNumber), 10, 36, 2)

End Function

Private Function modifyName(inputString As String) As String

    Dim currentCount As Long
    Dim strLength As Long
    Dim strArray(36) As String
    Dim integerX As Long
    Dim outputString As String
    Dim tempString As String

    currentCount = 0
    strLength = Len(inputString)
    outputString = ""

    While currentCount <= 35
        strArray(currentCount) = ""
        currentCount = currentCount + 1
    Wend

    currentCount = 1

    While currentCount <= strLength

        tempString = Mid(inputString, currentCount, 1)
        integerX = getCharIndex(tempString)
        strArray(integerX) = strArray(integerX) & tempString
        currentCount = currentCount + 1

    Wend

    currentCount = 0

    While currentCount <= 35
        outputString = outputString & strArray(currentCount)
        currentCount = currentCount + 1
    Wend
    modifyName = outputString

End Function

Private Function generateHashNo(inputString As String) As Long

    Dim thisCharIndex As Long
    Dim currentCount As Long
    Dim currentTotal As Long

    thisCharIndex = 0
    currentTotal = 0
    currentCount = 1

    While currentCount <= Len(inputString)
        thisCharIndex     =     getCharIndex_Extended(Mid(inputString,
currentCount, 1))
        currentTotal = generateTotal(currentTotal, thisCharIndex)
        currentCount = currentCount + 1
    Wend
    generateHashNo = currentTotal

End Function

Private Function generateTotal(total As Long, charIndex As Long) As
Long

    Dim currentCount As Long
    Dim currentTotal As Double
    Dim currentIndex As Double
    Dim F1 As Long
    Dim F2 As Long
    Dim k1 As Long
    Dim k2 As Long
    Dim multiplicand As Double
    Dim dividend As Double

    Dim intResponse As Integer

    currentCount = 1
    F1 = 117
    F2 = 0
    k1 = 39
    k2 = 7

    While currentCount <= 32
```

```
        currentTotal = CDbl(total)

        multiplicand = CDbl(charIndex) * 16
        multiplicand = makeOverflow(multiplicand)
        dividend = (charIndex - (charIndex Mod 32)) / 32
        dividend = makeOverflow(dividend)

        currentTotal  =  currentTotal  +  (multiplicand  +  dividend)  +
charIndex + F2 + k1
        currentTotal = makeOverflow(currentTotal)
        total = CLng(currentTotal)

        F2 = F2 + F1

        currentIndex = CDbl(charIndex)

        multiplicand = CDbl(total) * 16
        multiplicand = makeOverflow(multiplicand)
        dividend = (total - (total Mod 32)) / 32
        dividend = makeOverflow(dividend)

        currentIndex  =  currentIndex  +  (multiplicand  +  dividend)  +
total + F2 + k2
        currentIndex = makeOverflow(currentIndex)
        charIndex = CLng(currentIndex)

        currentCount = currentCount + 1
    Wend
    generateTotal = total

End Function

Private  Function  obfuscateString(inputString  As  String,  obfBase  As
Long, totalChars As Long, outputChars As Long) As String

    Dim currentCount As Long
    Dim multiplyNumber As Long
    Dim multiplyDouble As Double
    Dim currentTotal As Long
    Dim doubleTotal As Double
    Dim strLength As Long
    Dim outputString As String
    Dim charIndex As Long

    currentCount = 0
    multiplyNumber = 1
    currentTotal = 0
    strLength = Len(inputString)
    outputString = ""

    While currentCount < strLength

        doubleTotal = CDbl(currentTotal)
        doubleTotal     =     doubleTotal     +     multiplyNumber     *
getCharIndex(Mid(inputString, strLength - currentCount, 1))
```

```
        doubleTotal = makeOverflow(doubleTotal)
        currentTotal = CDbl(doubleTotal)

        multiplyDouble = CDbl(multiplyNumber)
        multiplyDouble = multiplyDouble * obfBase
        multiplyDouble = makeOverflow(multiplyDouble)
        multiplyNumber = CLng(multiplyDouble)

        currentCount = currentCount + 1

    Wend

    currentCount = 0

    While currentCount < outputChars

        charIndex = currentTotal Mod totalChars
        If charIndex < 0 Then
            charIndex = totalChars + charIndex
        End If
        If charIndex > (totalChars - 1) Then
            charIndex = charIndex - totalChars
        End If
        outputString = getChar(charIndex) & outputString
        currentTotal = (currentTotal - charIndex) / totalChars
        currentCount = currentCount + 1

    Wend

    obfuscateString = outputString

End Function

Private Function getCharIndex(inputString As String) As Long

    Dim currentIndex As Long
    Dim thisChar As String
    currentIndex = 1

    While currentIndex <= Len(strAlpNum)

        thisChar = Mid(strAlpNum, currentIndex, 1)
        If StrComp(thisChar, inputString) = 0 Then
            getCharIndex = currentIndex - 1
            Exit Function
        End If
        currentIndex = currentIndex + 1

    Wend
    getCharIndex = 1

End Function

Private Function getCharIndex_Extended(inputString As String) As Long

    Dim currentIndex As Long
    Dim thisChar As String
    currentIndex = 1

    While currentIndex <= Len(strAlpNum2)

        thisChar = Mid(strAlpNum2, currentIndex, 1)
        If StrComp(thisChar, inputString) = 0 Then
            getCharIndex_Extended = currentIndex - 1
            Exit Function
        End If
        currentIndex = currentIndex + 1

    Wend
    getCharIndex_Extended = 1

End Function

Private Function getChar(inputInteger As Long) As String

    getChar = Mid(strAlpNum, inputInteger + 1, 1)

End Function

Private Function makeOverflow(inputNumber As Double) As Double

    While inputNumber > 2147483647 Or inputNumber < -2147483647
        If inputNumber > 2147483647 Then
            inputNumber = inputNumber - 4294967296#
        Else
            inputNumber = inputNumber + 4294967296#
        End If
    Wend
    makeOverflow = inputNumber

End Function
```

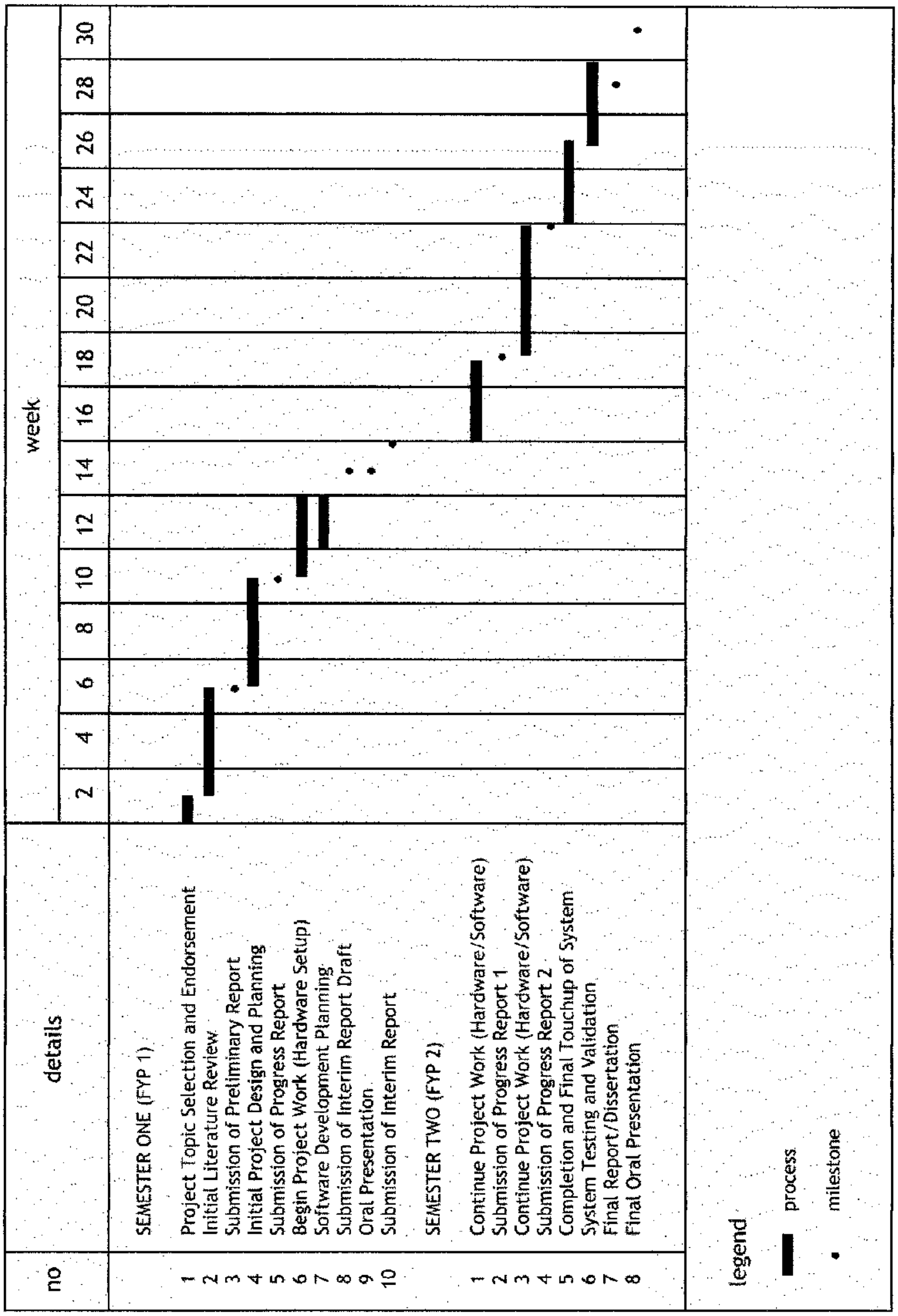
Appendix C: Client Software Source Code (Inventory)

```
'Initiate variables
Dim userName As String
Dim userMatric As String
Dim userActive As String
Dim userPassword As String
Dim userComponents(99) As Boolean
Dim userQuantity(99) As Boolean
Dim componentPicked As Boolean
Dim accountActive As Boolean
Dim itemsAdded As Integer
Dim itemsExisting As Integer
Dim originalItems As Integer

'Define data type for the inventory database
Private Type AccountRecord
    itemIndex As Long
    itemDate As String * 10
    itemStatus As Boolean
    itemName As String * 50
    itemQuantity As Long
End Type

Dim AccountData As AccountRecord
Dim AllAccountData As AccountRecord
Dim AccountFile&
Dim AccountRecordLength&
Dim totalAccountRecords&

Private Sub btnActivate_Click()

    Load frmVerify
    frmVerify.Show

End Sub

Private Sub btnCheckout_Click()

    Dim newComponent As String
    Dim newCount As Integer
    Dim currentLine As String
    Dim currentComponent As String
    Dim currentCount As Integer
    Dim componentLoop As Integer
    Dim deleteIndex As Integer
    Dim fields() As String

    'Sets the name of the borrowed component + quantity

    If chkOthers.Value = 0 Then
        newComponent = listCom.List(listCom.ListIndex)
    ElseIf chkOthers.Value = 1 Then
        newComponent = txtComponentName.Text
    End If
    newCount = CInt(txtQuantity.Text)

    'Check: quantity should be a number
    If IsNumeric(txtQuantity.Text) Then

        'Has the user selected a component to add?
        If    componentPicked    Or    (chkOthers.Value    =    1    And
StrComp(txtComponentName.Text, "") <> 0) Then

            componentLoop = 0
            deleteIndex = -1

            'Check if the component already exists
            For componentLoop = 0 To listBasket.ListCount - 1
                currentLine = listBasket.List(componentLoop)

                fields() = Split(currentLine, " x ")
                currentComponent = fields(0)
                currentCount = CInt(fields(1))

                'If component already exists in the basket
                If (StrComp(currentComponent, newComponent) = 0) Then

                    'Mark for deletion
                    deleteIndex = componentLoop
                    newCount = newCount + CInt(currentCount)

                End If

            Next componentLoop

            'Remove existing entry
            If Not deleteIndex = -1 Then
                listBasket.RemoveItem (deleteIndex)
                itemsAdded = itemsAdded - 1
            End If

            'Add to component basket and reset quantity amount
            listBasket.AddItem newComponent & " x " & CStr(newCount)
            itemsAdded = itemsAdded + 1
            txtQuantity.Text = "1"

        Else
            showErrorMsg ("Please select a component first!")
        End If
    Else
        showErrorMsg ("Please enter a numeric quantity.")
    End If

End Sub


Private Sub btnConfirm_Click()

    Dim confirmMsg As String
    Dim finalConfirm As Integer
    Dim fields() As String
    Dim itemEntry As String
    Dim itemName As String
    Dim itemQuantity As String

    'Confirm selections
    confirmMsg = "Are you sure you want to add these " & itemsAdded & "
items to your account?"
    finalConfirm = MsgBox(confirmMsg, vbOKCancel + vbQuestion, "Confirm
Items")

    'Move items once selection is confirmed
    If finalConfirm = 1 And listBasket.ListCount > 0 Then

        'For each item in the basket
        For basketCount = 0 To listBasket.ListCount - 1

            itemEntry = listBasket.List(basketCount)
            fields() = Split(itemEntry, " x ")
            itemName = fields(0)
            itemQuantity = fields(1)

            itemsExisting = itemsExisting + 1
            lvInventory.ListItems.Add , , CStr(Date)
            lvInventory.ListItems(itemsExisting).SubItems(1) = "Borrowed"
            lvInventory.ListItems(itemsExisting).SubItems(2) = itemName
            lvInventory.ListItems(itemsExisting).SubItems(3)          =
itemQuantity

        Next basketCount

    End If

    'Clear all values
    listBasket.Clear
    chkOthers.Value = 0
    txtComponentName.Text = ""

End Sub

Private Sub btnRemoveItem_Click()

    'If an item is selected, remove it
    If Not listBasket.ListIndex = -1 Then

        listBasket.RemoveItem listBasket.ListIndex
        itemsAdded = itemsAdded - 1

    End If

End Sub

Private Sub btnReturn_Click()

    Dim currentIndex As Integer
    Dim accountInfo As String

    currentIndex = 0
    accountInfo = ""

    'Send the entire list back to the server
    While currentIndex < itemsExisting

        currentIndex = currentIndex + 1
        accountInfo             =              accountInfo              &
lvInventory.ListItems(currentIndex).Text & ",,"
        accountInfo             =              accountInfo              &
lvInventory.ListItems(currentIndex).SubItems(1) & ",,"
        accountInfo             =              accountInfo              &
lvInventory.ListItems(currentIndex).SubItems(2) & ",,"
        accountInfo             =              accountInfo              &
lvInventory.ListItems(currentIndex).SubItems(3) & "++"

    Wend

    'Send the command line
    pendingAction = "updateAccount"
    passVariable1 = userMatric
    passVariable2 = accountInfo
    passVariable3 = CStr(itemsExisting)

    'Delete everything
    Close AccountFile
    Kill "TempRecord.db"
    lvInventory.ListItems.Clear

    'Return to main menu
    frmInventory.Hide
    fmClient.Show
    Unload frmInventory

End Sub

Private Sub Form_Unload(Cancel As Integer)

    fmClient.Show

End Sub

Private Sub btnReturnItems_Click()

    Dim thisLine As String
    Dim thisLineSplit() As String
    Dim selectedLine As Integer
    Dim prevName As String
    Dim prevQuantity As String

    'Only perform an action if an item is selected
    If Not lvInventory.SelectedItem Is Nothing Then

        If
```

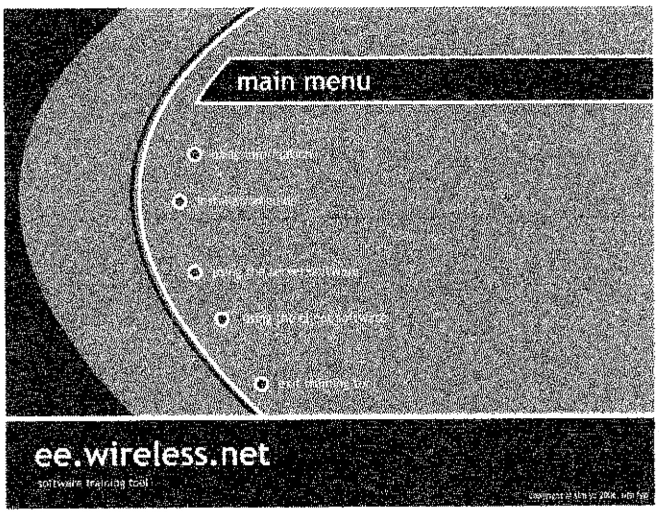
```
StrComp(lvInventory.ListItems(lvInventory.SelectedItem.Index).SubItems(1),
"Borrowed") = 0 Then
            prevName                                                    =
lvInventory.ListItems(lvInventory.SelectedItem.Index).SubItems(2)
            prevQuantity                                                =
lvInventory.ListItems(lvInventory.SelectedItem.Index).SubItems(3)

            lvInventory.ListItems.Remove (lvInventory.SelectedItem.Index)

            lvInventory.ListItems.Add , , CStr(Date)
            lvInventory.ListItems(itemsExisting).SubItems(1) = "Returned"
            lvInventory.ListItems(itemsExisting).SubItems(2) = prevName
            lvInventory.ListItems(itemsExisting).SubItems(3)            =
prevQuantity
        Else
            showErrorMsg ("That item has already been returned!")

        End If

    Else

        showErrorMsg ("Please select an item first!")

    End If

End Sub

Private Sub cboFamily_Click()

    'Grabs the list of components for this family
    Dim dataFileName As String
    Dim strFileLine As String
    componentPicked = False

    dataFileName = CStr(cboFamily.ItemData(cboFamily.ListIndex))
    dataFileName = "data\components\" & dataFileName & ".dat"

    If fileExists(dataFileName) Then

        listCom.Clear
        Open dataFileName For Input As #5

        'Load component list
        Do Until EOF(5)

            Line Input #5, strFileLine
            listCom.AddItem strFileLine

        Loop

        Close #5

    Else

        MsgBox ("File not found!")
    End If

End Sub

Private Sub Form_Load()

    Dim strFileLine As String
    Dim strComponents() As String
    Dim currentIndex As Integer

    componentPicked = False
    itemsAdded = 0
    itemExisting = 0

    userName = currentUserName
    userMatric = currentUserMatric
    userActive = currentUserActive

    lvInventory.SelectedItem = Nothing

    lblName.Caption = "Student Name: " & userName
    lblMatric.Caption = "Student Matric: " & userMatric

    'Displays the status of this account: active/inactive
    If (StrComp(userActive, "true") = 0) Then
        lblActive.Caption = "This account is active."
        btnActivate.Visible = False
        accountActive = True
    Else
        lblActive.Caption = "This account has NOT been activated!"
        btnActivate.Visible = True
        accountActive = False
    End If

    'prepare the random access file for the temporary account
    FileCopy "ClientRecord.db", "TempRecord.db"
    Open "ClientRecord.db" For Output As #2
    Close #2
    AccountFile = FreeFile
    AccountRecordLength = Len(AccountData)
    Open    "TempRecord.db"    For    Random    As    AccountFile    Len    =
AccountRecordLength
    totalAccountRecords = LOF(AccountFile) \ AccountRecordLength

    currentIndex = 0
    lvInventory.ListItems.Clear

    If LOF(AccountFile) = 0 Then

        itemsExisting = 0

    Else
        'Load component list
        While currentIndex < totalAccountRecords

            currentIndex = currentIndex + 1
            itemsExisting = currentIndex

            Get AccountFile, currentIndex, AccountData

            'display items in listview
            lvInventory.ListItems.Add , , Trim$(AccountData.itemDate)
            If AccountData.itemStatus = True Then
                lvInventory.ListItems(itemsExisting).SubItems(1)        =
"Returned"
            Else
                lvInventory.ListItems(itemsExisting).SubItems(1)        =
"Borrowed"
            End If
            lvInventory.ListItems(itemsExisting).SubItems(2)            =
Trim$(AccountData.itemName)
            lvInventory.ListItems(itemsExisting).SubItems(3)            =
Trim$(AccountData.itemQuantity)
        Wend

    End If

    originalItems = itemsExisting

    'call initialise controls
    InitControls

End Sub

Private Sub InitControls()

With cboFamily

        .AddItem "Capacitors - pF"
        .ItemData(0) = 0
        .AddItem "Capacitors - Ceramic"
        .ItemData(1) = 1
        .AddItem "Capacitors - Tantalium"
        .ItemData(2) = 2
        .AddItem "Capacitors - Electrolytic"
        .ItemData(3) = 3

        .AddItem "Connectors - WWrap"
        .ItemData(4) = 4
        .AddItem "Connectors - WWrapID"
        .ItemData(5) = 5
        .AddItem "Connectors - IC Socket"
        .ItemData(6) = 6
        .AddItem "Connectors - IDC Socket"
        .ItemData(7) = 7
        .AddItem "Connectors - D Connector"
        .ItemData(8) = 8
        .AddItem "Connectors - PCB Mounting"
        .ItemData(9) = 9
        .AddItem "Connectors - Terminals"
        .ItemData(10) = 10
        .AddItem "FYP Devices"
        .ItemData(11) = 11

        .AddItem "IC Analog - Linear"
        .ItemData(12) = 12
        .AddItem "IC Analog - Voltage Regulator"
        .ItemData(13) = 13
        .AddItem "IC Logic - 74 Series"
        .ItemData(14) = 14
        .AddItem "IC Logic - 40 Series"
        .ItemData(15) = 15

        .AddItem "Misc - PCB Prototyping Board"
        .ItemData(16) = 16
        .AddItem "Misc - 7 Segment/LED"
        .ItemData(17) = 17

        .AddItem "Resistors - Standard"
        .ItemData(18) = 18
        .AddItem "Resistors - Variable"
        .ItemData(19) = 19

        .AddItem "Semiconductors - Diodes"
        .ItemData(20) = 20
        .AddItem "Semiconductors - Transistors"
        .ItemData(21) = 21
        .AddItem "Semiconductors - Zener Diodes"
        .ItemData(22) = 22
        .AddItem "Semiconductors - MOSFET and IGBT"
        .ItemData(23) = 23
        .AddItem "Semiconductors - UJT, JFET & TRIAC"
        .ItemData(24) = 24

        .ListIndex = 18

    End With

    'display the components
    Dim dataFileName As String
    Dim strFileLine As String
    dataFileName = CStr(cboFamily.ItemData(cboFamily.ListIndex))
    dataFileName = "data\components\" & dataFileName & ".dat"
    If fileExists(dataFileName) Then
        Open dataFileName For Input As #5
        'load component list
        Do Until EOF(5)
            Line Input #5, strFileLine
            listCom.AddItem strFileLine
        Loop
        Close #5
    End If

End Sub

Private Function fileExists(ByVal filename As String)
```

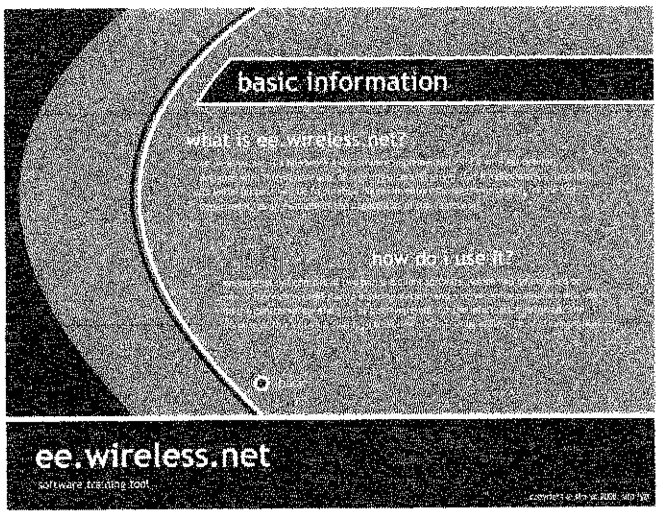
```
    'Checks if file exists
    Dim length As Long

    On Error GoTo FileDoesntExist

        length = FileLen(filename)
        fileExists = True

    Exit Function

FileDoesntExist:
        fileExists = False

End Function


Private Function showErrorMsg(errorEvent As String)

    'Shows an error message
    Dim intResponse As Integer

    intResponse = MsgBox(errorEvent, vbExclamation, "Error!")

End Function

Private Sub listCom_Click()

    componentPicked = True

End Sub
```

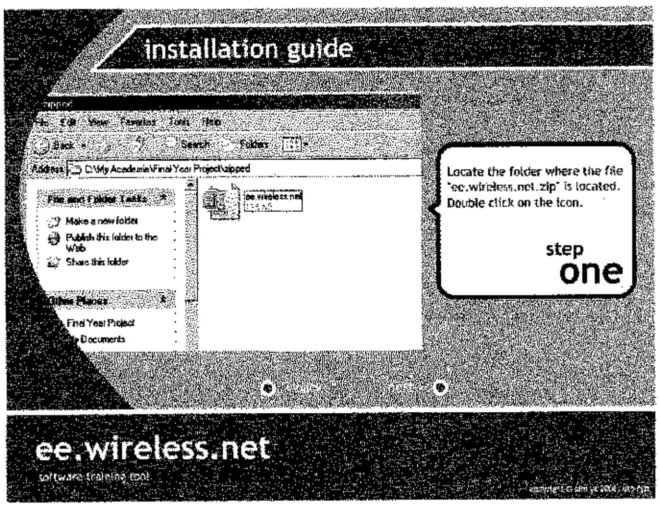
Appendix D: Gantt Chart



Appendix E: Software Training Tool Screenshots

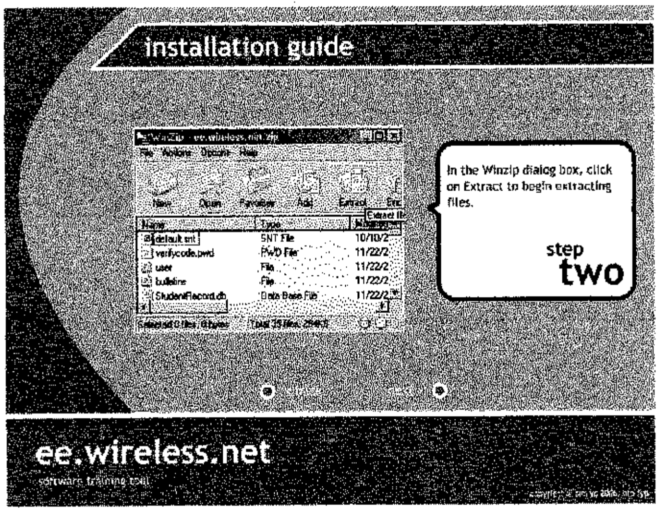






Locate the folder where the file "ee.wireless.net.zip" is located. Double click on the icon.

step one



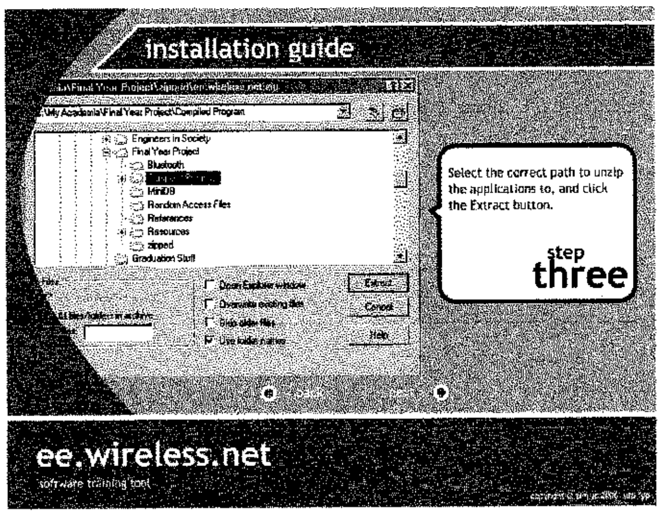
installation guide

In the Winzip dialog box, click on Extract to begin extracting files.

step two

WinZip - ee.wireless.net.zip
File Actions Options Help
New Open Favorites Add Extract
Name Type
default.snt SNT File 10/10/2
verifycode.pwd PWD File 11/22/2
user File 11/22/2
bulletine File 11/22/2
StudentRecord.db Data Base File 11/22/2

ee.wireless.net
software training tool

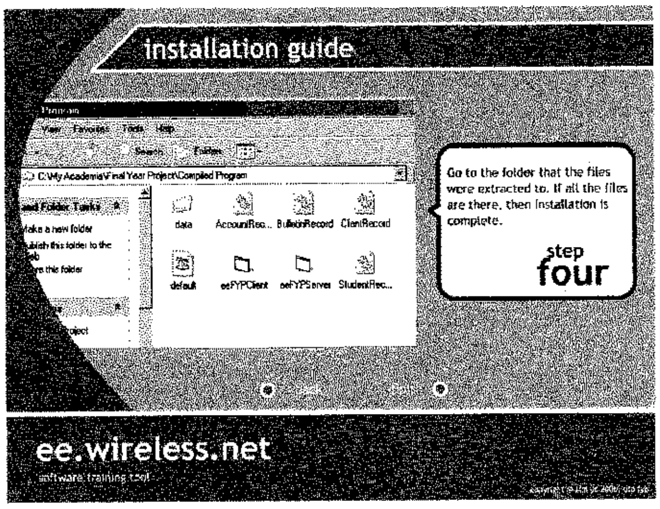


installation guide

Select the correct path to unzip the applications to, and click the Extract button.

step three

ee.wireless.net
software training tool

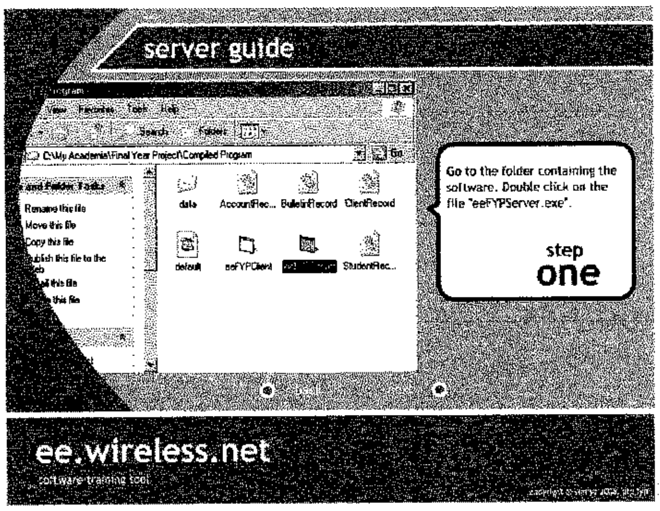


installation guide

Go to the folder that the files were extracted to. If all the files are there, then Installation is complete.

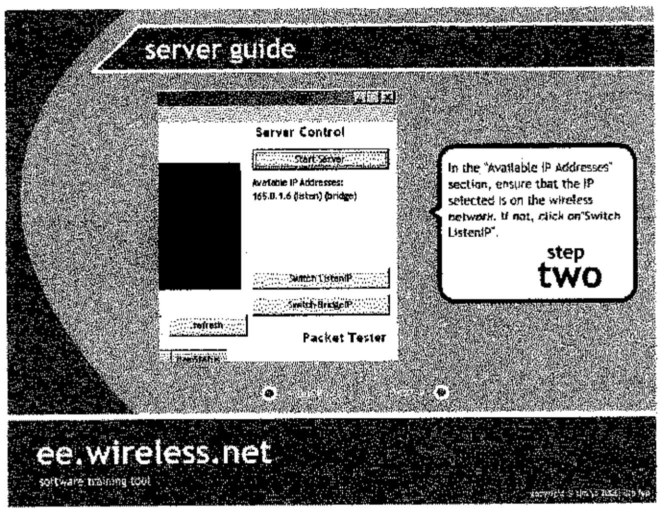
step four

ee.wireless.net
software training tool



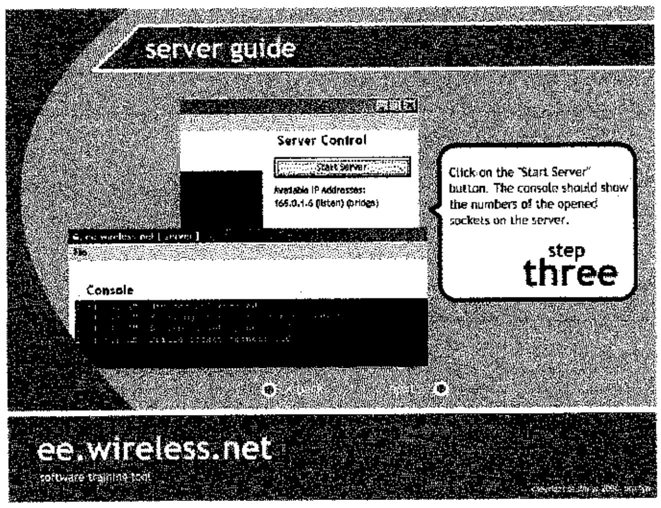
Go to the folder containing the software. Double click on the file "eeFYPServer.exe".

step one



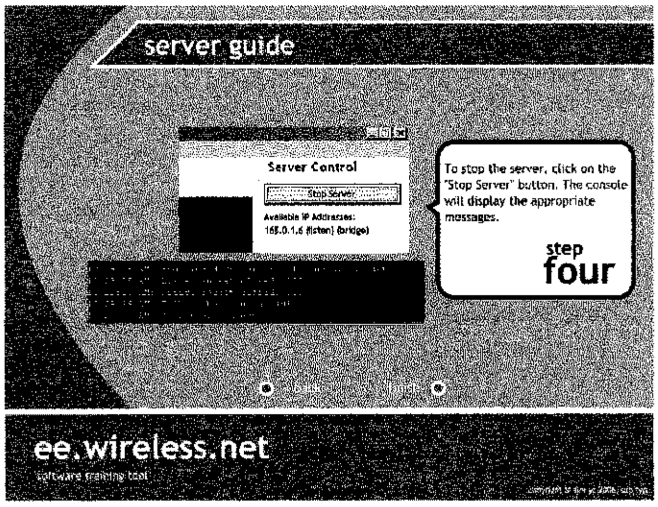
In the "Available IP Addresses" section, ensure that the IP selected is on the wireless network. If not, click on"Switch ListenIP".

step two



Click on the "Start Server" button. The console should show the numbers of the opened sockets on the server.

step three

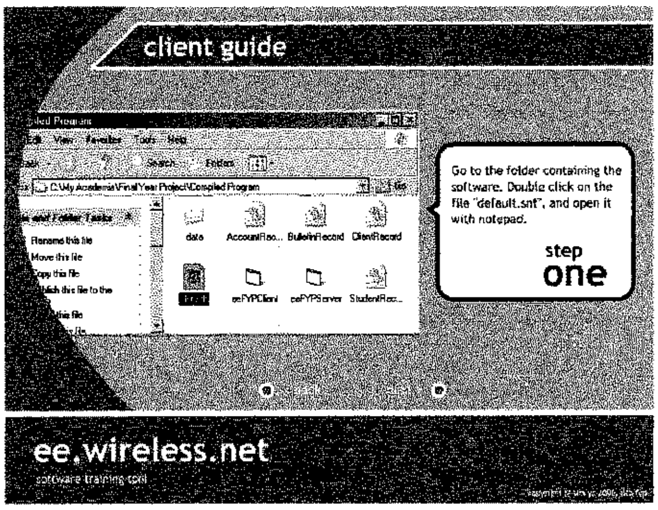


server guide

To stop the server, click on the "Stop Server" button. The console will display the appropriate messages.

step
four

ee.wireless.net
software training tool

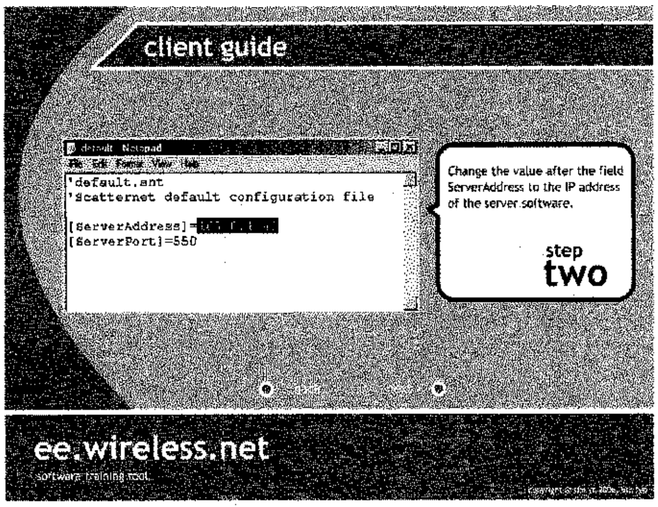


client guide

Go to the folder containing the software. Double click on the file "default.snt", and open it with notepad.

step
one

ee.wireless.net
software training tool



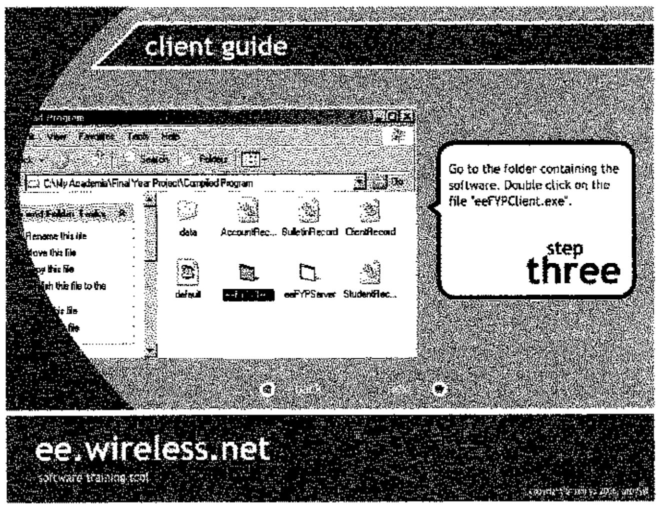
client guide

```
'default.snt
'Scatternet default configuration file

[ServerAddress]=[illegible]
[ServerPort]=550
```

Change the value after the field ServerAddress to the IP address of the server software.

step
two

ee.wireless.net
software training tool



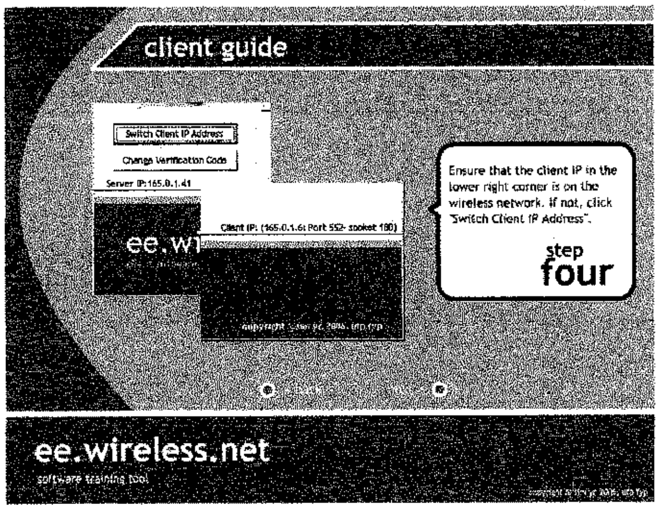
client guide

Go to the folder containing the software. Double click on the file "eeFYPClient.exe".

step three

ee.wireless.net

software training tool



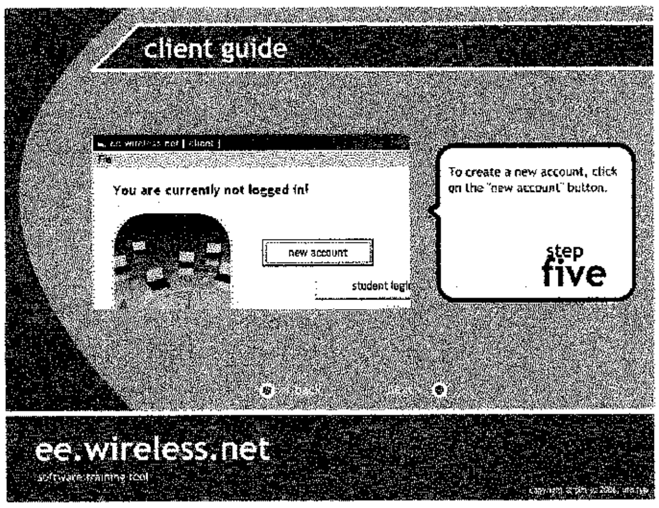
client guide

Ensure that the client IP in the lower right corner is on the wireless network. If not, click "Switch Client IP Address".

step four

ee.wireless.net

software training tool



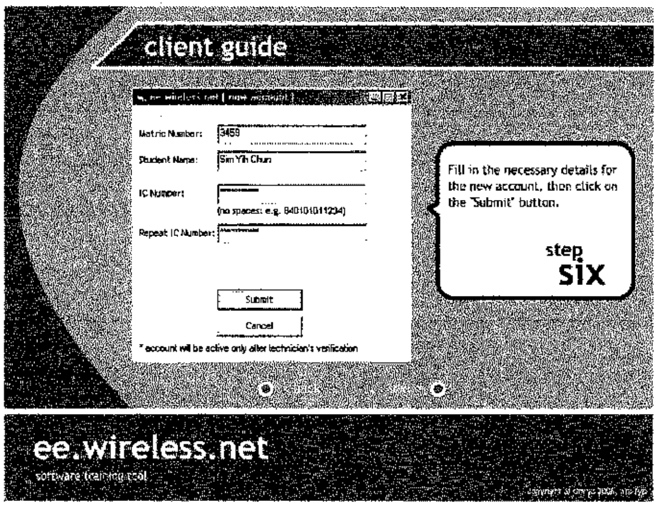
client guide

To create a new account, click on the "new account" button.

step five

ee.wireless.net

software training tool



client guide

ee.wireless.net [new account]

Matric Number: 3459

Student Name: Sim Yih Chun

IC Number:

(no spaces: e.g. 840101011234)

Repeat IC Number:

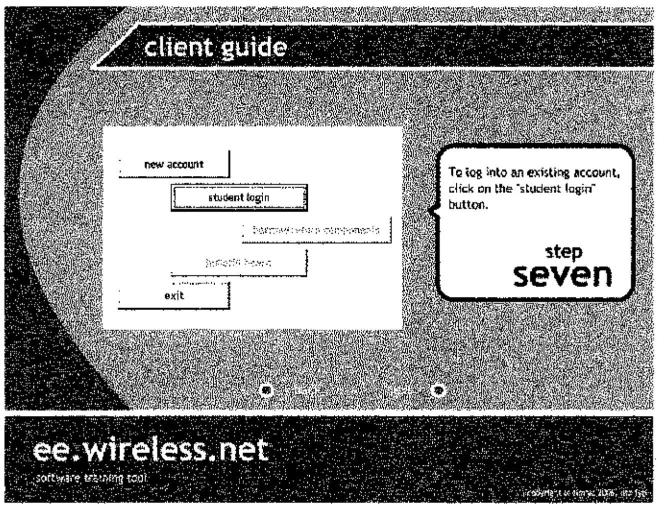
* account will be active only after technician's verification

Fill in the necessary details for the new account, then click on the "Submit" button.

step six

ee.wireless.net

software training tool



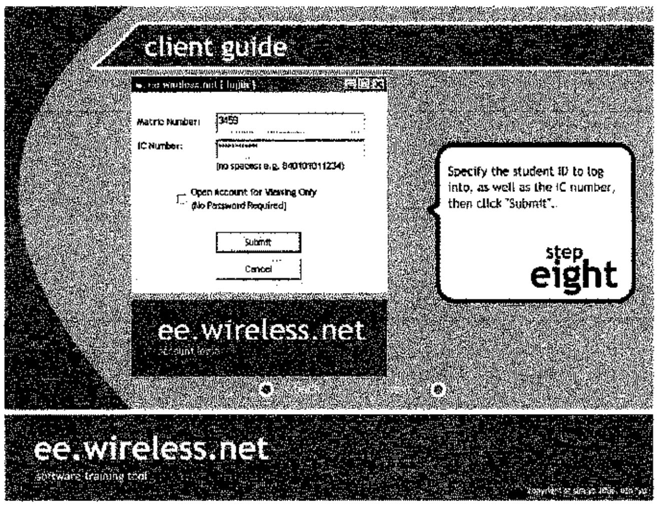
client guide

To log into an existing account, click on the "student login" button.

step seven

ee.wireless.net

software training tool



client guide

ee.wireless.net [login]

Matric Number: 3459

IC Number:

(no spaces: e.g. 840101011234)

Open Account for Viewing Only (No Password Required)

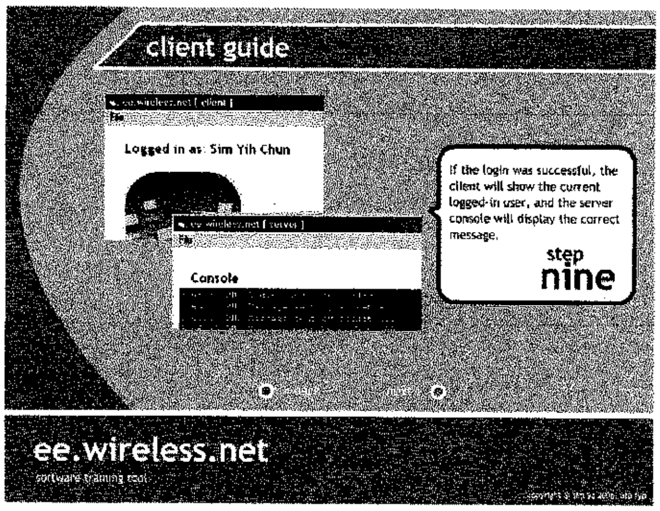
ee.wireless.net

Specify the student ID to log into, as well as the IC number, then click "Submit".

step eight

ee.wireless.net

software training tool



client guide

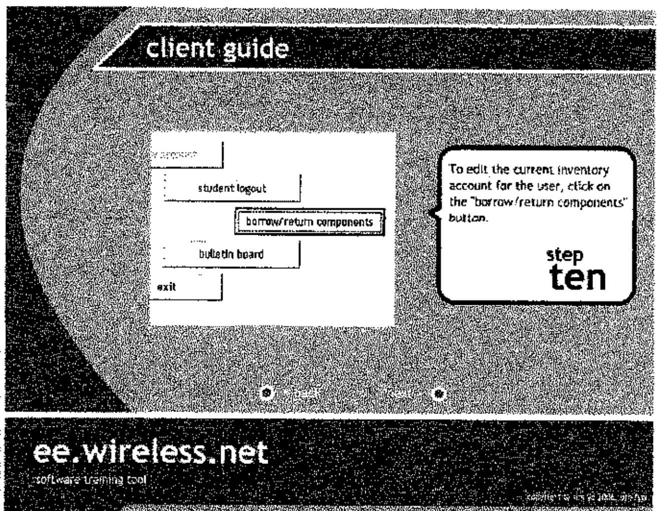
Logged in as: Sim Yih Chun

Console

If the login was successful, the client will show the current logged-in user, and the server console will display the correct message.

step
nine

ee.wireless.net
software training tool

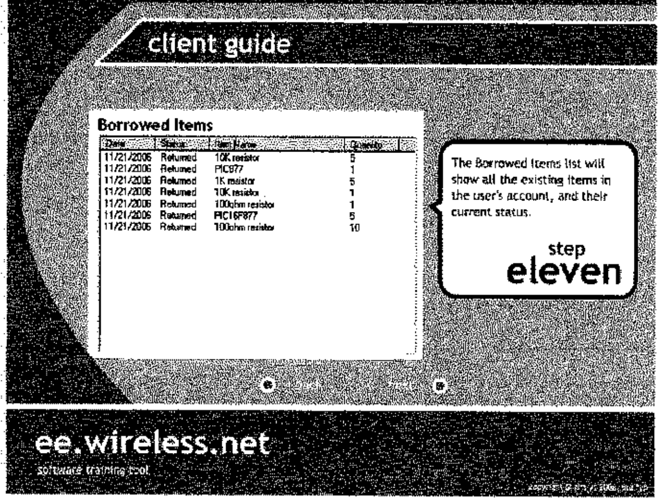


client guide

To edit the current inventory account for the user, click on the "borrow/return components" button.

step
ten

ee.wireless.net
software training tool



client guide

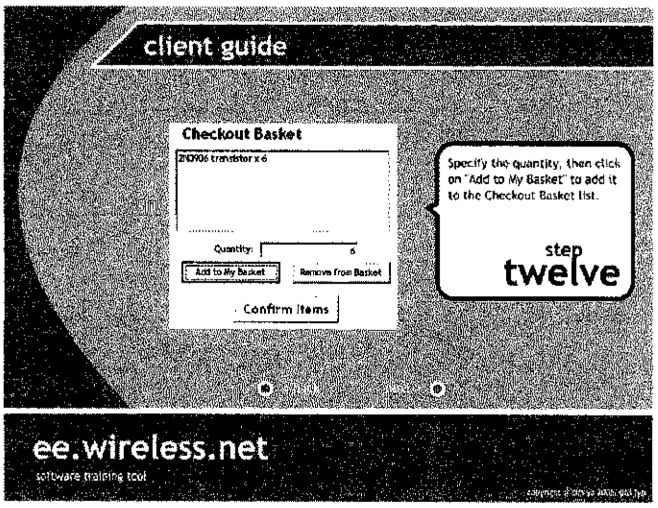
Borrowed Items

Date	Status	Item Name	Quantity
11/21/2006	Returned	10K resistor	5
11/21/2006	Returned	PIC877	1
11/21/2006	Returned	1K resistor	5
11/21/2006	Returned	10K resistor	1
11/21/2006	Returned	100ohm resistor	1
11/21/2006	Returned	PIC16F877	5
11/21/2006	Returned	100ohm resistor	10

The Borrowed Items list will show all the existing items in the user's account, and their current status.

step
eleven

ee.wireless.net
software training tool



client guide

Checkout Basket

2N3906 transistor x 6

Quantity: 6

Add to My Basket | Remove from Basket

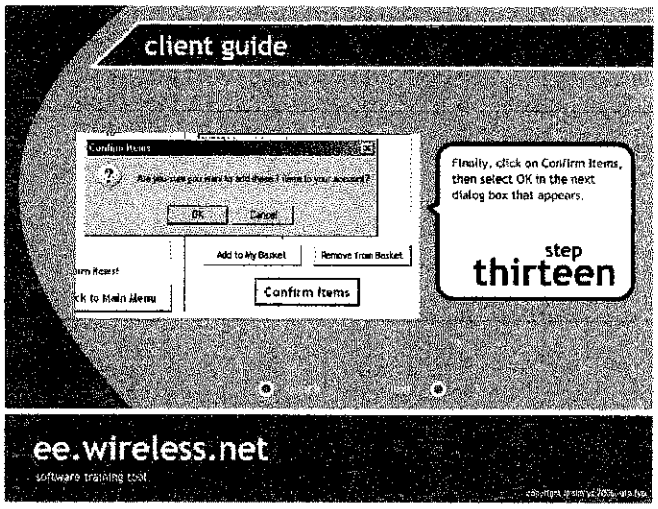
Confirm Items

Specify the quantity, then click on "Add to My Basket" to add it to the Checkout Basket list.

step **twelve**

ee.wireless.net

software training tool



client guide

Confirm Items

Are you sure you want to add these 1 items to your account?

OK | Cancel

Add to My Basket | Remove from Basket

Confirm Items

urn Items!

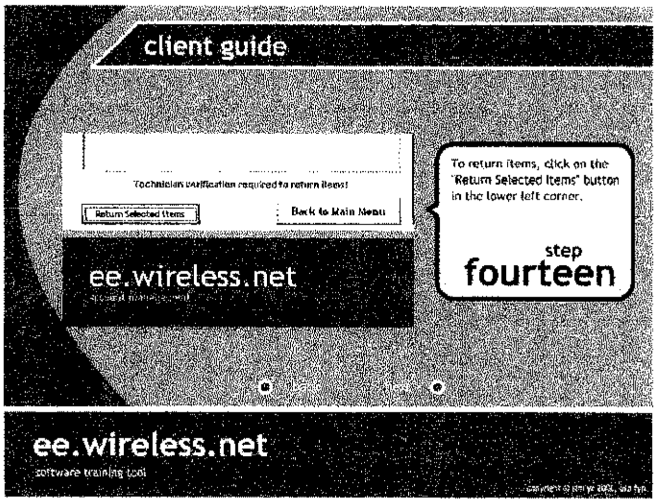
k to Main Menu

Finally, click on Confirm Items, then select OK in the next dialog box that appears.

step **thirteen**

ee.wireless.net

software training tool



client guide

Technician verification required to return items!

Return Selected Items | Back to Main Menu

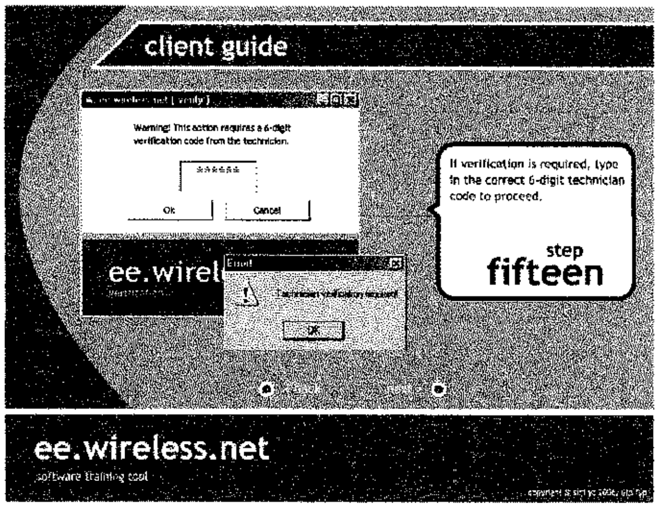
ee.wireless.net

To return items, click on the "Return Selected Items" button in the lower left corner.

step **fourteen**

ee.wireless.net

software training tool



client guide

Warning! This action requires a 6-digit verification code from the technician.

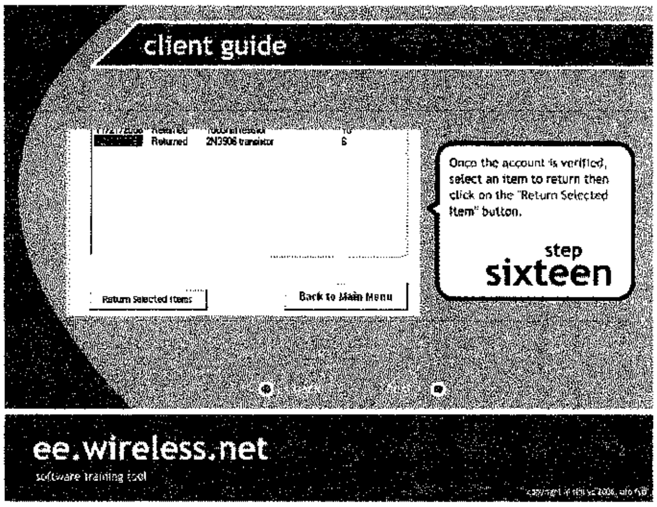
******

If verification is required, type in the correct 6-digit technician code to proceed.

step **fifteen**

ee.wireless.net

software training tool



client guide

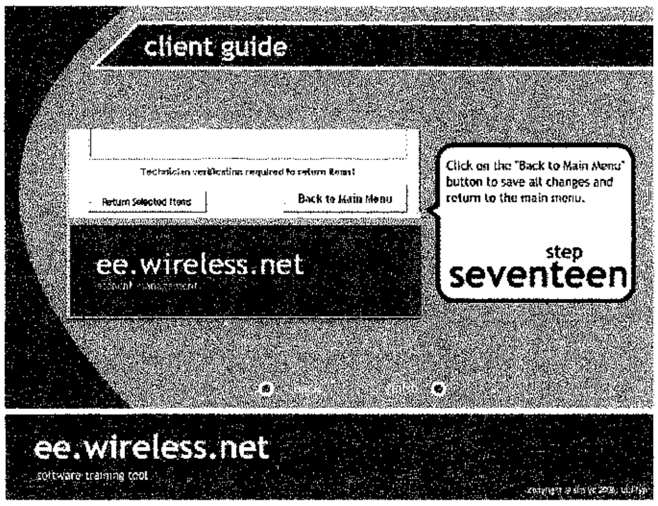
[illegible]	Returned	2N3906 transistor	6

Once the account is verified, select an item to return then click on the "Return Selected Item" button.

step **sixteen**

ee.wireless.net

software training tool



client guide

Technician verification required to return items!

Click on the "Back to Main Menu" button to save all changes and return to the main menu.

step **seventeen**

ee.wireless.net

software training tool

Appendix F: System Operation/User Manual

PROCESS	TECHNICIAN	CLIENT PC	WIRELESS CONNECTION	SERVER PC
Software Installation		1. Copy the compiled VB file eeFYPClient.exe to a folder on the PC. 2. Copy the file defaults.snt to the same folder. 3. Copy the folder data\components to the same folder. 4. To run the client software, double click on the eeFYPClient.exe icon.		1. Copy the compiled VB file eeFYPServer.exe to a folder on the PC. 2. If they are available, copy AccountRecord.db and StudentRecord.db to the PC. If these files do not exist, then they will be automatically created by the program. 3. To run the server software, double click on the eeFYPServer.exe icon.
Hardware Installation (Access Point)			1. Connect the access point to the power adapter. 2. Connect the access point to a PC using a standard RJ-45 Ethernet network cable. 3. Enter the address http://192.168.0.50 into an internet browser on the	

			PC. 4. The configuration wizard will start up. 5. Set the desired configuration settings, as detailed in section 3.5 of the report. 6. Disconnect the Ethernet cable. 7. With the antenna attached, turn the access point on to start the wireless network coverage.	
Hardware Installation (USB Adapter)		1. Insert the WiFi adapter's driver CD into the client PC. 2. Once the configuration wizard starts, begin the driver installation. 3. Continue clicking "Next" until the drivers are installed. 4. Restart the client PC. 5. Insert the Wireless G USB Adapter into the USB port of the PC.		1. Insert the WiFi adapter's driver CD into the server PC. 2. Once the configuration wizard starts, begin the driver installation. 3. Continue clicking "Next" until the drivers are installed. 4. Restart the server PC. 5. Insert the Wireless G USB Adapter into the USB port of the PC.

		6. Configure the IP address as detailed in section 3.5 of the report.		6. Configure the IP address as detailed in section 3.5 of the report.
Server Initialization				1. Start the server software. 2. The message “Winsock API Enabled” should be displayed in the console window. 3. Click on “start server”. 4. The software will display the ports that have been opened on the server PC. 5. To switch the active IP, click on the “switch ListenIP” button. 6. The active IP is marked by the “(listen)” suffix.
Client Initialization		1. Open the default.snt file with notepad, or any other text editor. 2. Edit the [ServerAddress] field to the correct server IP address. 3. Start the client software.		

Account Creation		1. Start the client software. 2. Click on the "new account" button. 3. Fill in all the appropriate values into the fields provided. 4. Click on submit.	5. The information will be assembled into a single packet and sent across the network to the server IP.	6. The packet received is processed to recover the original data. 7. AccountRecord.db is updated with a new entry.
Account Login		1. Start the client software. 2. Click on the "student login" button. 3. Enter the correct student ID and IC number. 4. Click on submit.		

		11. The client software acknowledges the login as authentic and opens access to other functions.	5. The information from the login is assembled into a single packet. 10. A packet is sent telling the client that the login was successful.	6. The packet is received and processed to extract the original information. 7. The software retrieves the correct user data. 8. The IC number of the retrieved account is compared to the user supplied number. 9. If the numbers are identical, the server sends a login successful message.
Borrow components		1. Start the client software. 2. Login as a user.		

		3. Click on the "borrow/ return components" button. 4. The inventory page for the current user will be shown. 5. A request is sent for the current user's existing data.		
				6. The account information for the user requesting the data is collected and assembled into a packet. 7. The packet is sent back to the client.
		8. The client software displays the existing inventory. 9. Additional components are selected by selecting them from the component list, then clicking the "add to my basket" button. 10. To change the amount, set the desired value in the Quantity text field. 11. Once the selection is		

		confirmed, click on the "Confirm Items" button. 12. To return to the main menu, click on the "Back to main menu" button.		13. The server updates the user's account data to reflect all changes.
Return components		1. Start the client software. 2. Login as a user. 3. Click on the "borrow/ return components" button. 4. The inventory page for the current user will be shown. 5. A request is sent for the current user's existing data.		6. The account information for the user requesting the data is collected and assembled into a packet. 7. The packet is sent back to the client.

		8. The client software displays the existing inventory.		
		9. Components are returned by selecting them and clicking on the “Return selected items” button.		
	10. Enter the correct verification code to authorize the transaction.			
		11. To return to the main menu, click on the “Back to main menu” button.		