UNIVERSITY OF ÇUKUROVA

INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

ABDÜLVAHAP SAYGIN

A Technique of Managing Clients via Internet

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA, 2000

ÇUKUROVA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

A TECHNIQUE OF MANAGING CLIENTS VIA INTERNET

ABDÜLVAHAP SAYGIN

YÜKSEK LİSANS TEZİ

ELEKTRİK-ELEKTRONİK ANA BİLİM DALI

Bu tez 04/09/2000 Tarihinde Aşağıdaki Jüri Üyeleri Tarafından Oybirliği/Oyçokluğu İle Kabul Edilmiştir.

Y.Doç.Dr.Ziya Gökalp ALTUN	Y.Doç.Dr.Murat AKSOY	Y.Doç.Dr.Sami ARICA
DANIŞMAN	ÜYE	ÜYE

Bu tez Enstitümüz Elektrik-Elektronik Anabilim Dalında hazırlanmıştır.

Kod No: 1761

T.C. ÇUKUROVA ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜ 1973

Prof.Dr. Melih BORAL
Enstitü Müdürü

Bu Çalışma Ç.Ü. Araştırma Fonu Tarafından Desteklenmiştir.
Proje No: FBE.2000.YL.106

CONTENTS **PAGE**

ÖZ

YÜKSEK LİSANS TEZİ

A TECHNIQUE OF MANAGING CLIENTS VIA INTERNET

Abdülvahap SAYGIN

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK – ELEKTRONİK ANABİLİM DALI

Danışman : Yrd. Doç. Dr. Ziya Gökalp Altun
Yıl :2000, Sayfa :123
Jüri : Yrd. Doç. Dr. Ziya Gökalp Altun
Yrd. Doç. Dr. Murat Aksoy
Yrd. Doç.Dr. Sami Arıca

Bu çalışmada internet aracılığıyla bilgisayar kontrol yöntemi önerilmektedir. Metod kullanımı için minimum gereksinim **sunucu** (server) ve **istemci** (client) olmak üzere iki adet internet bağlantılı bilgisayardır.

Sunucu bilgisayar, istemci tarafından gönderilen bilgileri işleyerek sonuçlarını ekran görüntüsü ile birlikte istemciye geri göndermektedir. İstemci, kumanda ve görme yetkilerine göre, bilgileri sunucudan alır. Önerilen metod birden fazla sunucu ve istemciden oluşan servis sistemlerine uygulanabilir.

Anahtar Kelimeler: Uzaktan Kontrol, Winsock, TCP/IP, Sunucu/İstemci.

ABSTRACT

MSc THESIS

A TECHNIQUE OF MANAGING CLIENTS VIA INTERNET

Abdülvahap SAYGIN

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
INSTITUTE OF NATURAL AND APPLIED SCIENCES
UNIVERSITY OF ÇUKUROVA

Supervisor : Assist. Prof. Dr. Ziya Gökalp Altun
Year :2000, Pages :123
Jury : Assist. Prof. Dr. Ziya Gökalp Altun
Assist. Prof. Dr. Sami Arıca
Assist. Prof. Dr. Murat Aksoy

In this study, a method of remote control of computers via internet is proposed. Minimum requirement to use the method is two computers, server and client, with internet connections

Server computer processes sent command by client and sends back relevant information with its screen snapshot to client. Client gets data due to permission of remote viewing and controlling of process. The proposed software can be applied to servicing systems, which are constructed with more than one server and one client.

Key Words: Remote Control, Winsock, TCP/IP, and Server/Client

ACKNOWLEDGMENTS

I would like to express my gratitude for Dr. Ziya Gökalp ALTUN and his wife for his great support, his unbelievable tolerance and encouragement with kindness and friendliness.

I would like to thank to Dr. Turgut İKİZ and Ümit Mücteba TOPALOĞLU for their support and friendship.

I also would like to thank my family members- I would like to thank my sister Derya SAYGIN especially- for their support, guidance and encouragement. I would like to express my great gratitude to my fiancé Hümeyra DEMİRBAŞ, for her confidence and love.

ACRONYMS

ADO	ActiveX® Data Object
ASCII	American Standard Code for Information Interchange
AUTO	Working with automatically
.bmp	Bitmap
.dll	Dynamic link Library
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
HTTPs	Hyper Text Transfer Protocol secure
IP	Internet Protocol
.jpeg(or jpg)	The Joint Pictures Experts Group
LAN	Local Area Network
.ocx	Open Compact Exchange
SQL	Structured Query Language
TCP	Transfer Control Protocol
UDP	User Datagram Protocol
VB	Visual Basic
Winsock	Windows Socket
Server	The computer which controls the process in a production unit
Client	The computer which will be connected to server via internet/intranet

LIST OF TABLES **PAGES**

LIST OF FIGURES **PAGES**

1. INTRODUCTION

Internet technology allows companies to overcome many of the physical constraints. For years, research has focused on ways to allow remote access via standard communication. With the growth of Internet, one finds more and more devices such as coffee machines, telescopes, manipulators, and mobile robots connected to it (Saucy and Mondana, 2000).

Capturing and sending server screen gives remote viewing features to client. Screen is captured as BMP and then converted to JPEG. Graphics, of which bits of information, well exceeds the volume of textual data (Stone, 1999). Compressing by JPEG reduces the file size range that is tolerable for internet delivery (Mintzer, 1999). Delivering time of picture file is less than 1 second, but depends on internet speed. Communications bandwidth and access time are changing rapidly, but they are the main bottleneck for image access (Stone, 1999).

Remote-control software is a program in a central or server computer that is used to control other computers (or their users) at a distance, either under the control of an administrator or at the request of the user. Although remote-control software existed before the World Wide Web (for remote diagnosis of computer problems and other purposes), the Web has essentially built a platform on which anyone can build a new remote-control application that can reach millions of computers and their users. Remote-control software can be viewed as one class of application furnished by application service provider.

Remote-control software can be divided into applications for use within a private network (such as an intranet) or for use on the public network. In a private network, remote-control software can be used to configure and administer all computers from a central point. On the public network, users can request such services as name lookup or arrange to have their files backed up automatically once a day. Remote-control software can also be used in a classroom system where one PC becomes the "master" of student computers, automatically reconfiguring them or turning them off at night.

Remote access is the ability to get access to a computer or a network from a remote distance. In corporations, people at branch offices, telecommuters, and people who are travelling may need access to the corporation's network. Home users get access to the Internet through remote access to an Internet service provider (Internet service provider). Dial-up connection through desktop, notebook, or handheld computer modem over regular telephone lines is a common method of remote access. Remote access is also possible using a dedicated line between a computer or a remote local area network and the "central" or main corporate local area network. A dedicated line is more expensive and less flexible but offers faster data rates. Integrated Services Digital Network is a common method of remote access from branch offices since it combines dial-up with faster data rates. wireless, cable modem, and Digital Subscriber Line technologies offer other possibilities for remote access.

A remote access server is the computer and associated software that is set up to handle users seeking access to network remotely. Sometimes called a communication server, a remote access server usually includes or is associated with a firewall server to ensure security and a router that can forward the remote access request to another part of the corporate network. A remote access server may include or work with a modem pool manager so that a small group of modems can be shared among a large number of intermittently present remote access users. A remote access server may also be used as part of a virtual private network.

The proposed system is applied a simple process, and tried (and succeed) to one server, two client system and also one client, two server system. However, if the system architecture is different from applied system, it can be applied by changing in server software.

Furthermore, one may have existing process control or wants to construct different process control system, of which code is unable to dissolve, and desires to use remote control part of proposal. In this case, it is possible with existing screen capture utility and adding capability of mouse and keyboard control of server (Topaloğlu, 2000). The result of this change in server software gives opportunity to control process and correct errors as if using server computer.

This work can be compared with the commercial software PcANYWHERE™ (SYMANTEC). Like in it, two computers are communicated bi-directional with each other. There are, also, other programs, generally illegal, to control the contrary computers. All these type of programs have two parts: server and client part. Client is the user who wish to control other computer, and the server is the controlled machine. For this work, Visual Basic™ 6.0 is used as the programming medium.

2. ESSENTIALS OF WINDOWS SOCKET (WINSOCK) AND TCP/IP

Protocols are system programs developed to allow cooperating computers to share resources across a network (gopher-chem.ucdavis.edu). These were developed by a community of researchers centered on the ARPAnet. The most accurate name for the set of protocols we are describing is the "Internet Protocol Suite". The term "Internet" applies to this entire set of networks, which are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. The Internet protocol documents are simply standards adopted by the Internet community for its own use.

Some of the most commonly used protocols are:

- TCP/IP
- NetBEUI
- X.25
- Xerox Network System (XNS™)
- IPX/SPX and NWLink
- APPC
- AppleTalk
- OSI Protocol Suit
- DECnet

TCP/IP (Transmission Control Protocol and Internet Protocol) is an industry standard protocol. Therefore this protocol is used for application. Appendix A consists of information about other protocols.

2.1.TCP/IP

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network. It was developed by a community of researchers centered on the ARPAnet. The most accurate name for the set of protocols we are describing is the "Internet protocol suite". TCP and IP are two of the protocols in this

suite. Because TCP and IP are the best known of the protocols, it has become common to use the term TCP/IP or IP/TCP to refer to the whole family.

The Internet is a collection of networks, including the Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The term "Internet" applies to this entire set of networks. The subset of them that is managed by the Department of Defense is referred to as the "DDN" (Defense Data Network). This includes some research-oriented networks, such as the Arpanet, as well as more strictly military ones (Because much of the funding for Internet protocol developments is done via the DDN organization, the terms Internet and DDN can sometimes seem equivalent). All of these networks are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. Officially speaking, the Internet protocol documents are simply standards adopted by the Internet community for its own use. More recently, U.S.A. Department of Defense issued a MILSPEC definition of TCP/IP. This was intended to be a more formal definition, appropriate for use in purchasing specifications. However most of the TCP/IP community continues to use the Internet standards. The MILSPEC version is intended to be consistent with it.

Whatever it is called, TCP/IP is a family of protocols. A few provide "low-level" functions needed for many applications. These include IP, TCP, and UDP. Others are protocols for doing specific tasks, e.g. transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly between minicomputers or mainframes. These machines had their own disks, and generally were self-contained. Thus the most important "traditional" TCP/IP services are:

- *File transfer*: The file transfer protocol (FTP) allows a user on any computer to get files from another computer, or to send files to another computer. Security is handled by requiring the user to specify a user name and password for the other computer. Provisions are made for handling file transfer between machines with different character set, end of line conventions, etc. This is not quite the same thing as more recent "network file system" or "netbios" protocols, which will be

described below. Rather, FTP is a utility that you run any time you want to access a file on another system. You use it to copy the file to your own system. You then work with the local copy.

- *Remote login*: The network terminal protocol (TELNET) allows a user to log in on any other computer on the network. You start a remote session by specifying a computer to connect to. From that time until you finish the session, anything you type is sent to the other computer. Note that you are really still talking to your own computer. But the telnet program effectively makes your computer invisible while it is running. Every character you type is sent directly to the other system. Generally, the connection to the remote computer behaves much like a dialup connection. That is, the remote system will ask you to log in and give a password, in whatever manner it would normally ask a user who had just dialed it up. When you log off of the other computer, the telnet program exits, and you will find yourself talking to your own computer. Microcomputer implementations of telnet generally include a terminal emulator for some common type of terminal.

- *Computer mail*: This allows to send messages to users on other computers. Originally, people tended to use only one or two specific computers. They would maintain "mail files" on those machines. The computer mail system is simply a way to add a message to another user's mail file. There are some problems with this in an environment where microcomputers are used. The most serious is that a micro is not well suited to receive computer mail. When one send mail, the mail software expects to be able to open a connection to the addressee's computer, in order to send the mail. If this is a microcomputer, it may be turned off, or it may be running an application other than the mail system. For this reason, mail is normally handled by a larger system, where it is practical to have a mail server running all the time. Microcomputer mail software then becomes a user interface that retrieves mail from the mail server.

These services should be present in any implementation of TCP/IP, except that micro-oriented implementations may not support computer mail. These traditional applications still play a very important role in TCP/IP-based networks. However more recently, the way in which networks are used has been changing. The

older model of a number of large, self-sufficient computers is beginning to change. Now many installations have several kinds of computers, including microcomputers, workstations, minicomputers, and mainframes. These computers are likely to be configured to perform specialized tasks. Although people are still likely to work with one specific computer, that computer will call on other systems on the net for specialized services. This has led to the "server/client" model of network services. A server is a system that provides a specific service for the rest of the network. A client is another system that uses that service. (Note that the server and client need not be on different computers. They could be different programs running on the same computer).

- *Network file systems*: This allows a system to access files on another computer in a somewhat more closely integrated fashion than FTP. A network file system provides the illusion that disks or other devices from one system are directly connected to other systems. There is no need to use a special network utility to access a file on another system. The computer simply thinks it has some extra disk drives. These extra "virtual" drives refer to the other system's disks. This capability is useful for several different purposes. It lets to put large disks on a few computers, but still give others access to the disk space. Aside from the obvious economic benefits, this allows people working on several computers to share common files. It makes system maintenance and backup easier. A number of vendors now offer high-performance diskless computers. These computers have no disk drives at all. They are entirely dependent upon disks attached to common "file servers".

- *Remote printing:* This allows to access printers on other computers as if they were directly attached. (The most commonly used protocol is the remote lineprinter protocol from Berkeley Unix).

- *Remote execution*: This allows requesting that a particular program be run on a different computer. This is useful when most of the work is on a small computer, but a few tasks require the resources of a larger system. There are a number of different kinds of remote execution. Some operate on a command-by-command basis. That is, requesting a specific command or set of commands should run on some specific computer. However there are also "remote procedure call"

systems that allow a program to call a subroutine that will run on another computer (There are many protocols of this sort. Berkeley Unix contains two servers to execute commands remotely: rsh and rexec. The man pages describe the protocols that they use).

- *Name servers:* In large installations, there are a number of different collections of names that have to be managed. This includes users and their passwords, names and network addresses for computers, and accounts. It becomes very tedious to keep this data up to date on all of the computers. Thus the databases are kept on a small number of systems. Other systems access the data over the network.

- *Terminal servers*: Many installations no longer connect terminals directly to computers. Instead they connect them to terminal servers. A terminal server is simply a small computer that only knows how to run telnet (or some other protocol to do remote login). If your terminal is connected to one of these, you simply type the name of a computer, and you are connected to it. Generally it is possible to have active connections to more than one computer at the same time. The terminal server will have provisions to switch between connections rapidly, and to notify you when output is waiting for another connection (Terminal servers use the telnet protocol, already mentioned. However any real terminal server will also have to support name service and a number of other protocols.).

- *Network-oriented window systems*: Until recently, high- performance graphics programs had to execute on a computer that had a bit-mapped graphics screen directly attached to it. Network window systems allow a program to use a display on a different computer. Full-scale network window systems provide an interface that lets to distribute jobs to the systems that are best suited to handle them, but still give a single graphically-based user interface. The most widely implemented window system is X.

Note that the list above is simply a sample of the sort of services available through TCP/IP. However it does contain the majority of the "major" applications. The other commonly used protocols tend to be specialized facilities for getting information of various kinds, such as who is logged in, the time of day, etc.

TCP/IP (Transmission [or Transport] Control Protocol and Internet Protocol) is an industry standard suite of protocols providing communication in a heterogeneous environment. In an addition, TCP/IP provides a routable, enterprise networking protocol and access to the Internet and its resources. It is a transport layer protocol that actually consists of several other protocols in a stack that operates at the session layer. Almost all networks support TCP/IP as a protocol (Microsoft Press, 1997).

It has become the standard protocol used for interoperability among many different types of computers. This interoperability is one of the primary advantages to TCP/IP. Almost all networks support TCP/IP as a protocol. TCP/IP also supports routing, and is commonly used as an internetworking protocol.

Because of its popularity, TCP/IP has become the TCP/IP suite include:

- SMTP (Simple Mail Transfer Protocol e-mail)
- FTP (File Transfer Protocol) For exchanging files among computers
- SNMP (Simple Network Management Protocol) Network Management.

Historically, there were two primary disadvantages of TCP/IP: its size and speed. TCP/IP is a relatively large protocol stack, which can cause problems in MS-DOS-based clients. However, on graphically user interface (GUI) based operating systems, such as Windows NT/2000 or Windows 98, the size is not an issue and speed is about the same as IPX.

2.2. Winsock and Properties

A WinSock control allows to be connected to a remote machine and exchange data using either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). Both protocols can be used to create client and server applications. Like the Timer control, the WinSock control doesn't have a visible interface at run time. Possible uses:

- Create a client application that collects user information before sending it to a central server.
- Create a server application that function as a central collection point for data from several users.
- Create a "chat" application.

For using output of the Ethernet socket, winsock.ocx and winsock.dll are used. This ocx and dll are very skillful and competent tools. Winsock is part of the Internet Control Productivity (ICP) pack. The ICP is currently being developed. It is planned to support as many Internet protocols as possible. ICP will provide the user with enhanced connectivity options. It will especially be beneficial to developers who need firewall support. The controls in the ICP project will provide support primarily for the widely in use SOCKS protocol. Winsock Properties are given in Appendix B.

2.2.1. Selecting A Protocol

When using the Winsock control, the first consideration is whether to use the TCP or the UDP protocol. The major difference between the two lies in their connection state:

- The TCP protocol control is a connection-based protocol, and is analogous to a telephone — the user must establish a connection before proceeding.
- The UDP protocol is a connectionless protocol, and the transaction between two computers is like passing a note: a message is sent from one computer to another, but there is no explicit connection between the two. Additionally, the network determines the maximum data size of individual sends.

The nature of the application are being created will generally determine which protocol select. Here are a few questions that may help select the appropriate protocol:

1. Will the application require acknowledgment from the server or client when data is sent or received? If so, the TCP protocol requires an explicit connection before sending or receiving data.

2. Will the data be extremely large (such as image or sound files)? Once a connection has been made, the TCP protocol maintains the connection and ensures the integrity of the data. This connection, however, uses more computing resources, making it more "expensive."

3. Will the data be sent intermittently, or in one session? For example, if an application that notifies specific computers is created when certain tasks have completed, the UDP protocol may be more appropriate. The UDP protocol is also more suited for sending small amounts of data.

2.2.2. Setting The Protocol

To set the protocol that application will use at design-time, on the Properties window, click Protocol and select either *sckTCPProtocol*, or *sckUDPProtocol*. The Protocol property in code, as shown below can be also set:

```
Winsock1.Protocol = sckTCPProtocol
```

2.2.3. Determining The Name of Computer

To connect to a remote computer, either its IP address or its "friendly name" must be known. The IP address is a series of three digit numbers separated by periods (xxx.xxx.xxx.xxx). In general, it's much easier to remember the friendly name of a computer. To find server computer's name,

Start\Control Panel\Network\Identification\Computer name box must be read.

Once clients computer name have been found , it can be used as a value for the RemoteHost property.

3. CONNECTING SERVER VIA INTERNET

In remote control terminology, a server computer, which is in Internet and/or Intranet networking system, means that a computer managed by other computers, which are called as clients. This work's aim is to control a server computer with a software which consists of four parts:

- Communication interface,
- Interface for talk between computers,
- Interface for controlling server's windows system,
- Capturing monitor and directing mouse of server.

3.1.Communication Interface

First task of this project is to provide connection between computers. The communication interface is written with Visual Basic™ 6.0. Interfaces are designed in two types : *TCP Connection Basis* and *UDP Connection Basis.*

3.1.1. TCP Connection Basics

When creating an application that uses the TCP protocol, it must be decided if its application will be a server or client. Creating a server means that the proposed application will "listen" on a designated port. When the client makes a connection request, the server can accept the request and thereby complete the connection, or refuse the request. If request is accepted and the connection is complete, then client and server can freely communicate with each other and both computers can stream data between themselves.

If a client application is being created, the server computer's name or IP address (*RemoteHost* property) must be known, as well as the port (*RemotePort* property) on which it will be "listening", then invokes the Connect method.

If a server application is being created, set a port (*LocalPort* property) on which to listen, and invoke the Listen method. When the client computer requests a connection, the *ConnectionRequest* event will occur. To complete the connection, invoke the Accept method within the *ConnectionRequest* event.

Once a connection has been made, either computer can send and receive data. To send data, invoke the *SendData* method. Whenever data is received, the *DataArrival* event occurs. The data is retrieved by invoking the *GetData* method within the *DataArrival* event.

The Winsock control is invisible to user, and provides easy access to TCP and UDP network services. Microsoft Access, Visual Basic™, Visual C++™, or Visual FoxPro™ developers can use it (CHAPPELL,D.). To write client or server applications do not need to be understood the details of TCP or to call low-level Winsock APIs. By setting properties and invoking methods of the control, a remote machine can be easily connected to a remote machine and exchange data in both directions. The Transfer Control Protocol allows creating and maintaining a connection to a remote computer. If a client applications are being created, the server computer's name or IP address (*RemoteHost* Property) must be known, as well as the port (*RemotePort* property) on which it will be "listening". Then invoke the Connect method.

3.1.2. UDP Connection Basics

The User Datagram Protocol (UDP) is a connectionless protocol. Unlike TCP operations, computers do not establish a connection. Also, a UDP application can be either a client or a server (Microsoft® Visual Basic® 5.0 Active X).

To transmit data, first set the client computer's *LocalPort* property. The server computer then needs only to set the *RemoteHost* to the Internet address of the client computer, and the *RemotePort* property to the same port as the client computer's *LocalPort* property, and invoke the *SendData* method to begin sending messages. The client computer then uses the *GetData* method within the *DataArrival* event to retrieve the sent messages.

3.2. Interface for Talk Between Computers

3.2.1. Password Interface

Two side of the net (server and client) are connected each other and managed by this interface. When the program runs, the window (fig.3.1) arises on client's screen.

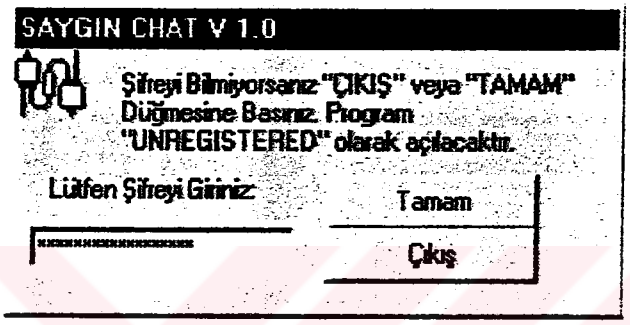

Figure 3.1. The Password Interface.

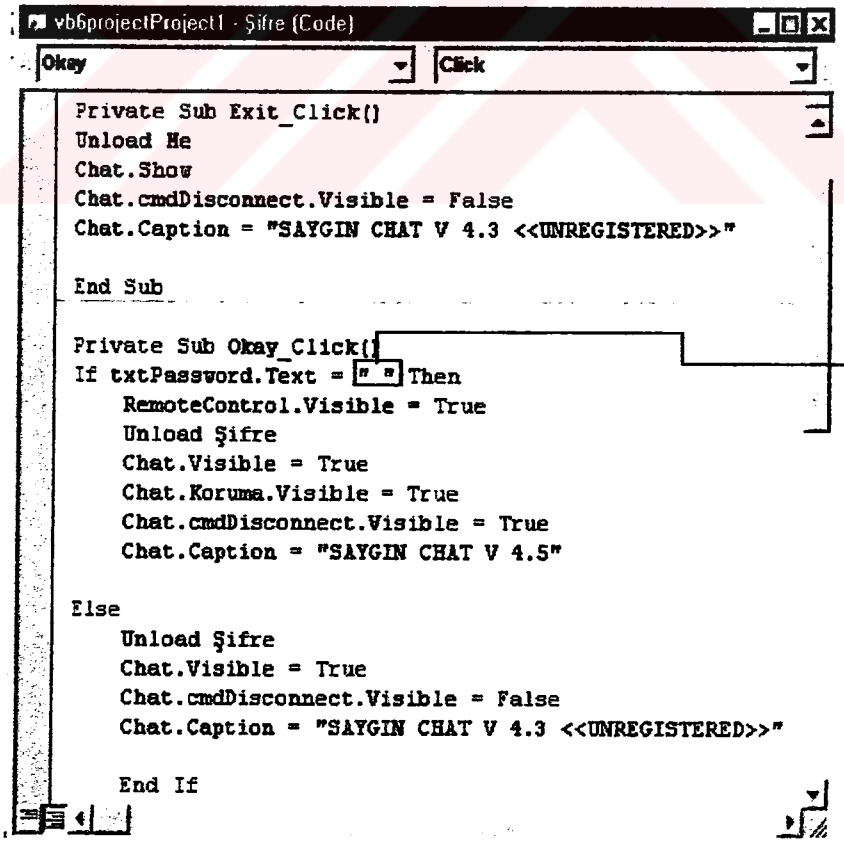

Figure 3.2. The Source Code of Password Interface

Source code of this window is shown on fig.3.2. If the "Çıkış" button (fig.3.1) is pressed, the part of the "Private Sub Exit_Click()" will be activated. The password form is unloaded and the chat form is loaded. If the "Tamam" button is pressed. the part of the "Private Sub Okay_Click()" function is activated. When the password is entered correctly, the first part of the "IF" loop is activated. The Main form is opened with all properties.If the wrong password is entered or pressed "Tamam" or "Çıkış" button, this program will be open as shown below:

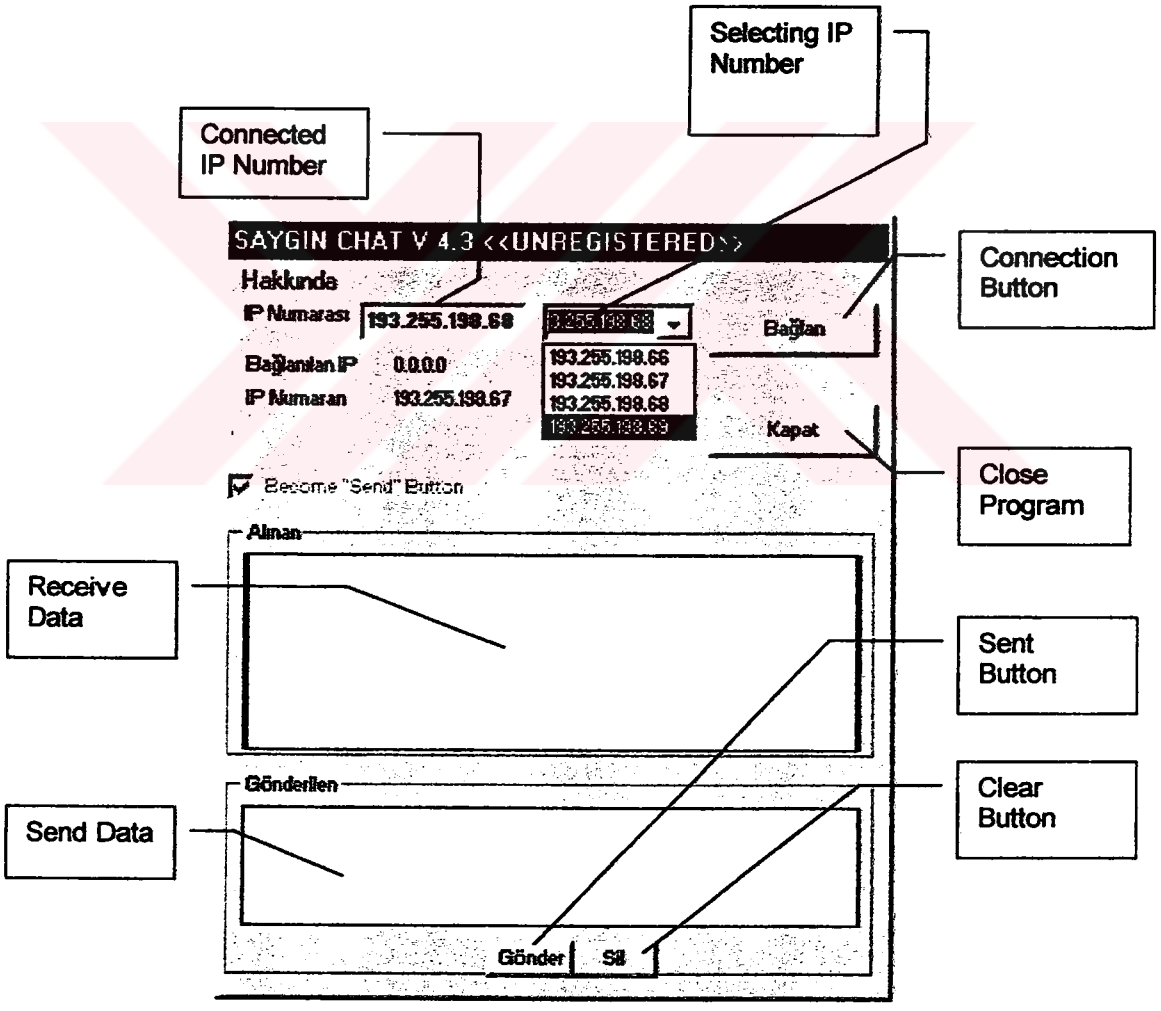

Figure 3.3. Connection Interface

3.2.2. Communication Interface with TCP Basis

The communication interface is shown in Figure 3.3. The same interface has been used for connection information and chat application with the user of connected computer.

Receive Data: The sent data from other client is captured and written in this box.

Send Data: Data would be sent from user, if the "Gönder" button were clicked. "Sil" button clears this box.

Connected IP: The IP would be connected which can either be written or selected from *Selecting IP Box.*

Connection Button ("Bağlan"): When clicked onto, the contrary computer will be connected.

Close Button ("Kapat"): Program will be closed when this button is clicked.

If the correct password is entered on the window shown in fig.3.1., and pressed "Tamam" button, the invisible control panel will be visible (Figure 3.4).

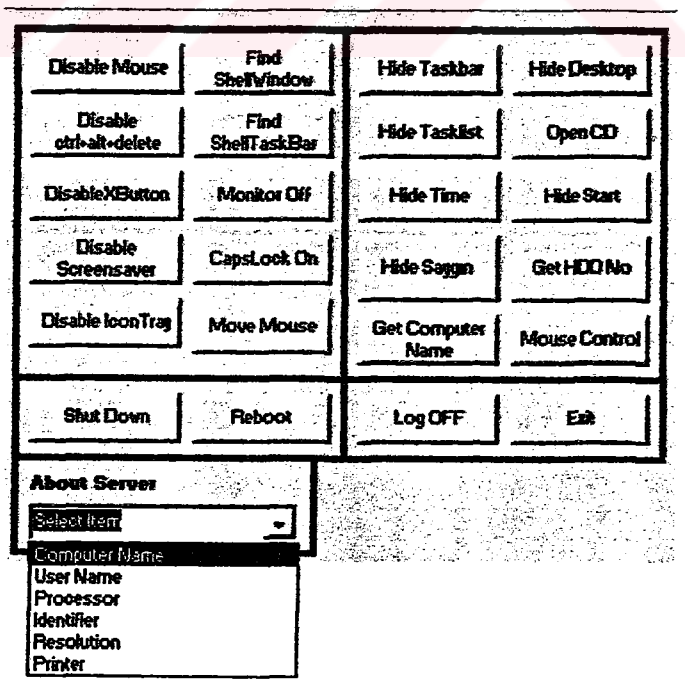

Figure 3.4. Connection and Managing Interface

The algorithm for program is given in figure 3.5.

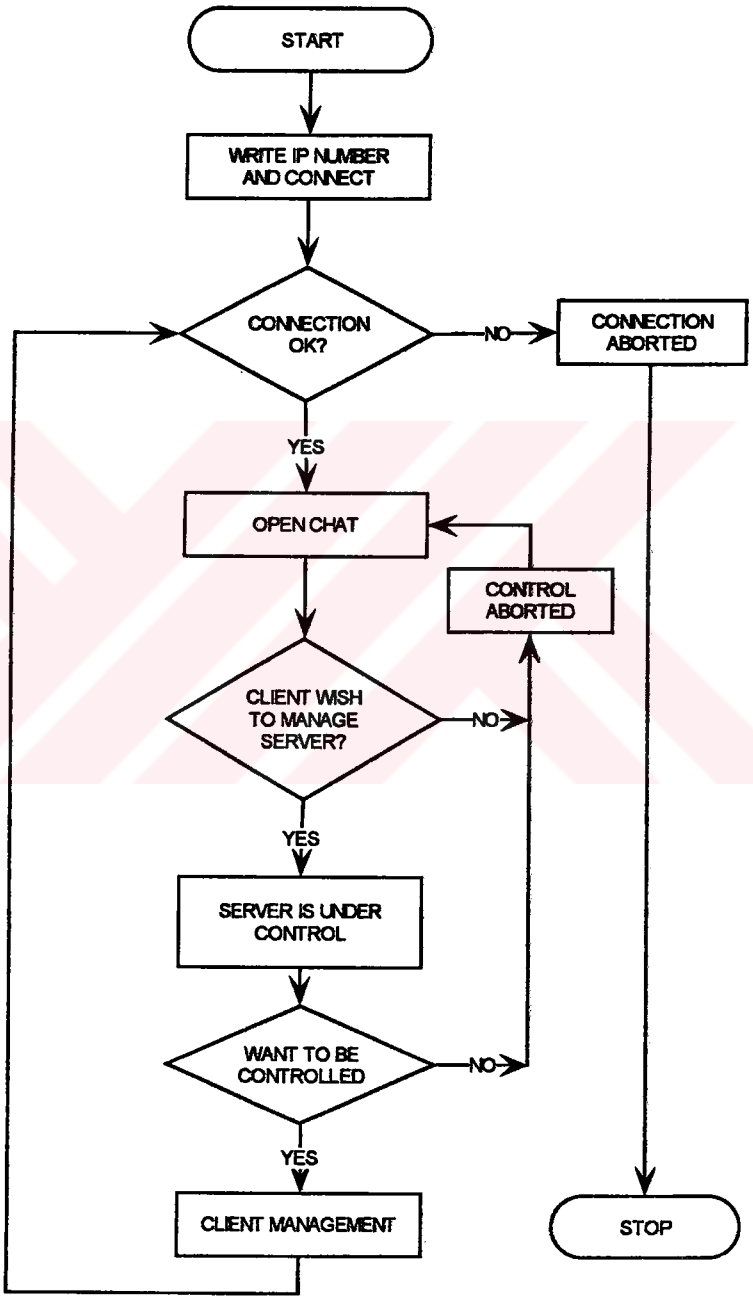

Figure 3.5. Algorithm of communication and management

The standard connection code is as shown below. First, the local and remote ports has to be defined:

```
Private Sub Form_Load()
tcpSERVER.LocalPort = 1001
tcpCLIENT.RemotePort = 1001
tcpSERVER.Listen
Şifre.Visible = True
End Sub
```

Both local port and remote port are selected at address 1001 and the program listens the address port 1001.

```
Private Sub cmdConnection_Click()
If txtIP.Text = lblClientIP.Caption Then     'to prevent the user to control
cmdConnection.Enabled = False                'the client computer (himself)
End If
tcpCLIENT.RemoteHost = txtIP.Text
tcpCLIENT.Connect
cmdDisconnect.Enabled = True
cmdConnection.Enabled = False
txtOutput.Clear
If TConnect1 = 0 Then
  txtOutput.AddItem txtIP.Text & "Numaralı bilgisayara bağlanmaya
çalışıyorum. Lütfen bekleyiniz"
 ElseIf TConnect1 = 1 Then
  txtOutput.AddItem " Bağlantı isteğiniz" _
  & txtIP.Text & " numaralı bilgisayar tarafından kabul edilmiştir."
  txtOutput.AddItem "Artık konuşabilirsiniz"
End If;End Sub
```

The variable "TConnect1" controls whom push the "Connect" button first. If TConnect1 is equal to '0', client will be connected to the server. If it is '1' then server will be connected to client. That is, this part of program decides who pressed button first.

When creating an application that uses the TCP protocol, it must be first decided if its application will be a server or a client. Creating a server means that application will "listen," on a designated port. When the client makes a connection request, the server can then accept the request and thereby complete the connection. Once the connection is complete, the client and server can freely communicate with each other.

```
Private Sub tcpServer_ConnectionRequest _
(ByVal requestID As Long)
    ' Check if the control's State is closed. If not,
    ' close the connection before accepting the new
    ' connection.
    If tcpServer.State <> sckClosed Then _
    tcpServer.Close
    ' Accept the request with the requestID
    ' parameter.
    tcpServer.Accept requestID
End Sub

Private Sub txtSendData_Change()
    ' The TextBox control named txtSendData
    ' contains the data to be sent. Whenever the user
    ' types into the  textbox, the  string is sent
    ' using the SendData method.
    tcpServer.SendData txtSendData.Text
End Sub

Private Sub tcpServer_DataArrival _
```

```
(ByVal bytesTotal As Long)
        ' Declare a variable for the incoming data.
        ' Invoke the GetData method and set the Text
        ' property of a TextBox named txtOutput to
        ' the data.
        Dim strData As String
        tcpServer.GetData strData
        txtOutput.Text = strData
End Sub
```

The procedures above create a simple server application. However, to complete the scenario, a client application must be also created a client application. The following code creates a simple client-server application.

```
Private Sub Form_Load()

        ' The name of the Winsock control is tcpClient.
        ' Note: to specify a remote host, you can use
        ' either the IP address (ex: "121.111.1.1") or
        ' the computer's "friendly" name, as shown here.
        tcpClient.RemoteHost = "RemoteComputerName"
        tcpClient.RemotePort = 1001
End Sub

Private Sub cmdConnect_Click()
        ' Invoke the Connect method to initiate a
        ' connection.
        tcpClient.Connect
End Sub
Private Sub txtSendData_Change()
        tcpClient.SendData txtSend.Text
```

```
End Sub
Private Sub tcpClient_DataArrival _
(ByVal bytesTotal As Long)
        Dim strData As String
        tcpClient.GetData strData
        txtOutput.Text = strData
End Sub
```

Accepting More than One Connection Request: The basic server outlined above accepts only one connection request. However, it is possible to accept several connection requests using the same control by creating a control array. In that case, there is no need to close the connection, but simply create a new instance of the control (by setting its Index property), and invoking the Accept method on the new instance.

The code given below assumes there is a Winsock control on a form named *sckServer*, and its Index property has been set to 0. In the Declarations section, a module-level variable *intMax* is declared. In the form's Load event, *intMax* is set to 0, and the *LocalPort* property for the first control in the array is set to 1001. Then the Listen method is invoked on the control, making it the "listening control. As each connection request arrives, the code tests to see if the Index is 0 (the value of the "listening" control). If so, the listening control increments *intMax*, and uses that number to create a new control instance. The new control instance is then used to accept the connection request.

```
Private intMax As Long
Private Sub Form_Load()
        intMax = 0
        sckServer(0).LocalPort = 1001
        sckServer(0).Listen
End Sub

Private Sub sckServer_ConnectionRequest _
```

```
(Index As Integer, ByVal requestID As Long)
    If Index = 0 Then
        intMax = intMax + 1
        Load sckServer(intMax)
        sckServer(intMax).LocalPort = 0
        sckServer(intMax).Accept requestID
        Load txtData(intMax)
    End If
End Sub
```

3.2.3. Communication Interface with UDP Basics

Creating a UDP application is even simpler than creating a TCP application because the UDP protocol doesn't require an explicit connection. In the TCP application above, one Winsock control must explicitly be set to "listen," while the other must initiate a connection with the Connect method.

In contrast, the UDP protocol doesn't require an explicit connection. To send data between two controls, three steps must be completed (on both sides of the connection):

1. Set the RemoteHost property to the name of the other computer.
2. Set the RemotePort property to the LocalPort property of the second control.
3. Invoke the Bind method specifying the LocalPort to be used.

Because both computers can be considered "equal" in the relationship, it could be called a peer-to-peer application. To demonstrate this, the code below creates a "chat" application that allows two people to "talk" in real time to each other:

```
Private Sub Form_Load()
        ' The control's name is udpPeerA
        With udpPeerA
                ' IMPORTANT: be sure to change the RemoteHost
                ' value to the name of your computer.
                .RemoteHost= "PeerB"
                .RemotePort = 1001   ' Port to connect to.
                .Bind 1002      'Bind to the local port.
        End With
        frmPeerB.Show          ' Show the second form.
End Sub

Private Sub txtSend_Change()
        ' Send text as soon as it's typed.
        udpPeerA.SendData txtSend.Text
End Sub

Private Sub udpPeerA_DataArrival _
(ByVal bytesTotal As Long)
        Dim strData As String
        udpPeerA.GetData strData
        txtOutput.Text = strData
End Sub

Private Sub Form_Load()
        ' The control's name is udpPeerB.
        With udpPeerB
                ' IMPORTANT: be sure to change the RemoteHost
                ' value to the name of your computer.
                .RemoteHost= "PeerA"
                .RemotePort = 1002   ' Port to connect to.
```

```
            .Bind 1001          ' Bind to the local port.
        End With
    End Sub

    Private Sub txtSend_Change()
        ' Send text as soon as it's typed.
        udpPeerB.SendData txtSend.Text
    End Sub

    Private Sub udpPeerB_DataArrival _
    (ByVal bytesTotal As Long)
        Dim strData As String
        udpPeerB.GetData strData
        txtOutput.Text = strData
    End Sub
```

As shown in the preceding code, the Bind method must be invoked when creating a UDP application. The Bind method "reserves" a local port for use by the control. For example, when the control is bound to port number 1001, no other application can use that port to "listen" on. This may come in useful if another application from using that port wish to be prevented. The Bind method also features an optional second argument. If there is more than one network adapter present on the machine, the *LocalIP* argument allows specifying which adapter to use.

4. MANAGING OF CLIENT VIA INTERNET

Important part of this project is to communicate and manage with client(s). The aim of communication is to give permission to the authorized person for managing the computer. A user (client) can also control more than one server.

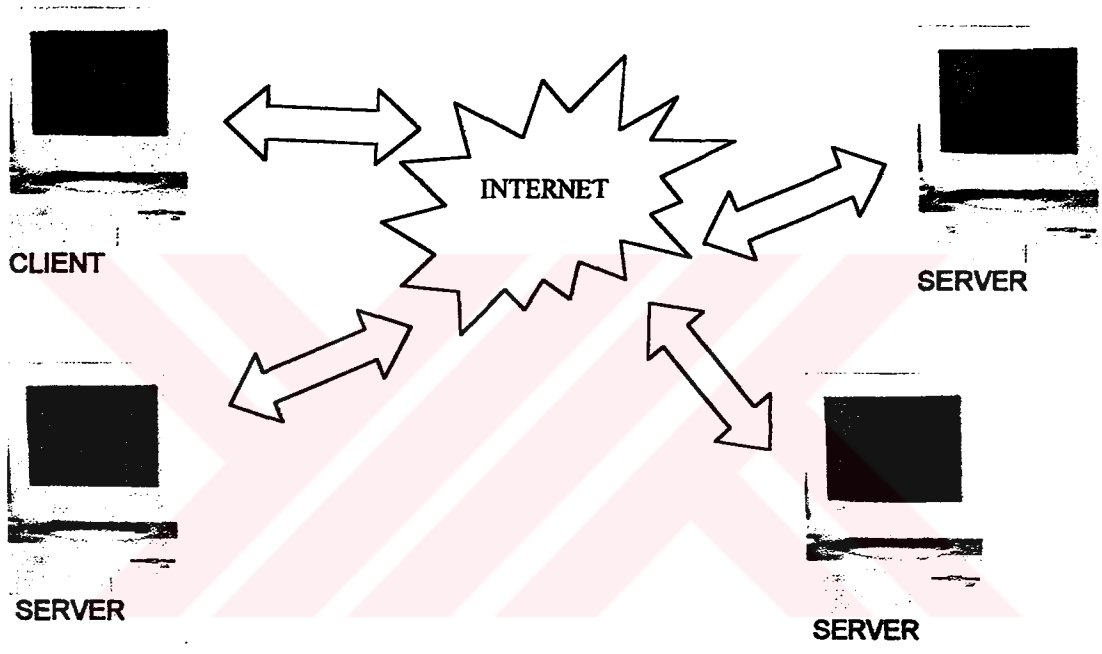

Figure 4.1. Client and server settling

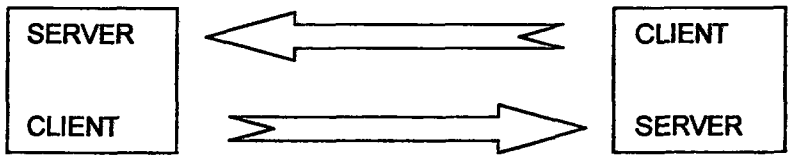

Figure 4.2. A user can be both client and server

4.1. Connecting for Chat

In chat section, server user and client user can talk each other in same time. There two text box that receive and send parts are included for chat. When any button is pushed that is any character is sent, the character is seemed to be in textbox. At the same time, the chat function is activated and the program immediately checks for remote connection. The remote command cannot mix chat character.

```
Private Sub tcpSERVER_DataArrival(ByVal bytesTotal As Long)
Dim strdata, tString As String
tcpSERVER.GetData tString
strdata = Vahap1.Encrypt(tString, 12)
If Koruma.Text = "ziya" Then
  txtOutput.AddItem "RECV:" & strdata
  txtOutput.ListIndex = txtOutput.ListCount - 1
Else
  If strdata = "œæhide mouse" Then
    RemoteOCX.DisableMouse = True
  ElseIf strdata = "œæshow mouse" Then
    RemoteOCX.DisableMouse = False
  ElseIf strdata = "œæhide taskbar" Then
    RemoteOCX.HideTaskBar = True
  ElseIf strdata = "œæshow taskbar" Then
    RemoteOCX.HideTaskBar = False
  ElseIf strdata = "œæhide ctrlaltdel" Then
    RemoteOCX.DisableCTRLALTDEL = True
  ElseIf strdata = "œæshow ctrlaltdel" Then
    RemoteOCX.DisableCTRLALTDEL = False
  ElseIf strdata = "œæhide xbutton" Then
    RemoteOCX.DisableXButton = True
  ElseIf strdata = "œæshow xbutton" Then
```

```
    RemoteOCX.DisableXButton = False
ElseIf strdata = "æædisable screensaver" Then
    Disable1.DisableScreenSaver = True
ElseIf strdata = "ææenable screensaver" Then
    Disable1.DisableScreenSaver = False
ElseIf strdata = "æhide saygınchat" Then
    RemoteOCX.HideShowWindow Chat.hWnd, True
    RemoteOCX.HideShowWindow RemoteControl.hWnd, True
ElseIf strdata = "æshow saygınchat" Then
    RemoteOCX.HideShowWindow Chat.hWnd, False
    RemoteOCX.HideShowWindow RemoteControl.hWnd, False
ElseIf strdata = "æhide desktop" Then
    RemoteOCX.HideDesktop = True
ElseIf strdata = "æshow desktop" Then
    RemoteOCX.HideDesktop = False
ElseIf strdata = "ælogoff" Then
    RemoteOCX.LogOff = True
ElseIf strdata = "æmove mouse" Then
    RemoteOCX.MouseMove 120, 120 = True
ElseIf strdata = "æunmove mouse" Then
    RemoteOCX.MouseMove 120, 120 = False
ElseIf strdata = "æshutdown" Then
    RemoteOCX.Shutdown = True
ElseIf strdata = "æfind shelltaskbar" Then
    RemoteOCX.FindShellTaskBar
ElseIf strdata = "æfind shellwindow" Then
    RemoteOCX.FindShellWindow
ElseIf strdata = "æhide start" Then
    Disable1.DisableStartButton = True
ElseIf strdata = "æshow start" Then
    Disable1.DisableStartButton = False
```

```
ElseIf strdata = "ææreboot" Then
  RemoteOCX.Reboot = True
ElseIf strdata = "æhide tasklist" Then
  RemoteOCX.RunningAsService = True
ElseIf strdata = "æshow tasklist" Then
  RemoteOCX.RunningAsService = False
ElseIf strdata = "æhide time" Then
  Disable1.DisableTrayClock = True
ElseIf strdata = "æshow time" Then
  Disable1.DisableTrayClock = False
ElseIf strdata = "ædisable icontray" Then
  Disable1.DisableIconTray = False
ElseIf strdata = "æenable icontray" Then
  Disable1.DisableIconTray = True
ElseIf strdata = "æhddno" Then
  txtHDDNo = SeriNoAl("c:\")
  tcpCLIENT.SendData (Vahap1.Decrypt(txtHDDNo.Text, 12))
ElseIf strdata = "æcomputername" Then
  Dim NameOfPC As String
  NameOfPC = GetComputer
  txtComputerName.Text = NameOfPC
  tcpCLIENT.SendData (Vahap1.Decrypt(txtComputerName.Text, 12))
ElseIf strdata = "æopencd" Then
  CDControl.EjectCD
ElseIf strdata = "æclosecd" Then
  CDControl.CloseCD
ElseIf strdata = "æcapslockon" Then
  GetKeyboardState kbArray
  kbArray.kbByte(VK_CAPITAL) = 1
  SetKeyboardState kbArray
  Label1 = IIf(CapsLock() = 1, "On", "Off")
```

```
ElseIf strdata = "ææcapslockoff" Then
    GetKeyboardState kbArray
    kbArray.kbByte(VK_CAPITAL) = 0
    SetKeyboardState kbArray
    Label1 = IIf(CapsLock() = 1, "On", "Off")
ElseIf strdata = "ææmonitoroff" Then
    Dim Ret As Long
    Ret = SendMessage(Chat.hWnd, WM_SYSCOMMAND,
SC_MONITORPOWER, 0&)
ElseIf strdata = "ææmonitoron" Then
    Dim Ret1 As Long
    Ret1 = SendMessage(Chat.hWnd, WM_SYSCOMMAND,
SC_MONITORPOWER, -1&)
ElseIf strdata = "ææcomputernamereg" Then
'Computer Name
RegEdit1.hKey = HKEY_LOCAL_MACHINE
    RegEdit1.SubKeyPath =
"System\CurrentControlset\Control\ComputerName\ComputerName"
    RegEdit1.ValueName = "ComputerName"
    RegEdit1.GetValue
    tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "ææusername" Then
  'User Name
RegEdit1.hKey = HKEY_LOCAL_MACHINE
    RegEdit1.SubKeyPath = "Network\Logon"
    RegEdit1.ValueName = "username"

    RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "ææprocessor" Then
    'Processor
```

```
On Error Resume Next
RegEdit1.hKey = HKEY_LOCAL_MACHINE
RegEdit1.SubKeyPath =
"Hardware\Description\System\CentralProcessor\0"
RegEdit1.ValueName = "Identifier"
RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "æævendoridentifier" Then
'Vendor Identifier
On Error Resume Next
RegEdit1.hKey = HKEY_LOCAL_MACHINE
RegEdit1.SubKeyPath =
"Hardware\Description\System\CentralProcessor\0"
RegEdit1.ValueName = "VendorIdentifier"
RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "ææresolution" Then
'resolution
On Error Resume Next
RegEdit1.hKey = HKEY_CURRENT_CONFIG
RegEdit1.SubKeyPath = "Display\Settings" = True
RegEdit1.ValueName = "Resolution" = True
RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "æædefaultprinter" Then
'Default Printer
On Error Resume Next
RegEdit1.hKey = HKEY_CURRENT_CONFIG
RegEdit1.SubKeyPath =
"System\CurrentControlSet\Control\Print\Printers"
```

```
            RegEdit1.ValueName = "Default"
            RegEdit1.GetValue
            tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
        ElseIf strdata = "ææSMTP" Then
            'SMTP Email Address
            RegEdit1.hKey = HKEY_CURRENT_USER
            RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
            RegEdit1.ValueName = "SMTP Email Address"
            RegEdit1.GetValue
            'SMTP Server
            RegEdit1.hKey = HKEY_CURRENT_USER
            RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
            RegEdit1.ValueName = "SMTP Server"
            RegEdit1.GetValue
            'POP3 User Name
            RegEdit1.hKey = HKEY_CURRENT_USER
            RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
            RegEdit1.ValueName = "POP3 User Name"
            RegEdit1.GetValue
            'Account Name
            RegEdit1.hKey = HKEY_CURRENT_USER
            RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
            RegEdit1.ValueName = "Account Name"

            RegEdit1.GetValue
            'Default Gateway
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
```

```
            RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans\0003"
            RegEdit1.ValueName = "DefaultGateway"
            RegEdit1.GetValue
            'IP Number
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans 0003"
            RegEdit1.ValueName = "IPAddress"
            RegEdit1.GetValue
            'IPMask
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans 0003"
            RegEdit1.ValueName = "IPMask"
            RegEdit1.GetValue
            'Name server
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans 0003"
            RegEdit1.ValueName = "NameServer1"
            RegEdit1.GetValue
            RegEdit1.ValueName = "NameServer2"
            RegEdit1.GetValue
          Else
            If MouseControl.Check2 Then
              MouseControl.Label4.Caption = strdata
            End If
            txtOutput.AddItem "RECV:" & strdata
            txtOutput.ListIndex = txtOutput.ListCount - 1
            RemoteControl.Show
```

```
        Call Activate
      End If
    End If
    End Sub
```

The server program as shown above can be distinguished management command and chat characters.

4.2. Managing Client User Computer

The server program has the capability of managing client user computer. If the defender password, which can be known only by manager(s), is entered, the server user cannot control client's computer. The control consists of:

- Hide Task Bar / Show Task Bar
- Disable Icon Tray / Enable Icon Tray
- Hide Start Button / Show Start Button
- Hide Tray Clock / Show Tray Clock
- Disable Ctrl+Alt+Delete function / Enable Ctrl+Alt+Delete function
- HideAll open windows / Show All open windows
- Run All open windows / Kill All open windows
- Shutdown Windows operating system
- Reboot Windows operating system
- Logoff Windows (does not close control program)
- Open CDRom / Close CDRom
- Get Hard Disk Number
- Get Computer Name
- Monitor ON / Monitor OFF
- Caps Lock ON/ Caps Lock OFF
- Get Computer Information

4.2.1. Hide Task Bar / Show Task Bar

When the "Hide Taskbar" button is clicked, the taskbar would be hidden. The source code is:

```
Private Sub Command3_Click()
If Command3.Caption = "Hide Taskbar" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææhide taskbar", 12)
   Command3.Caption = "Show Taskbar"
   Exit Sub
End If
If Command3.Caption = "Show Taskbar" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææshow taskbar", 12)
   Command3.Caption = "Hide Taskbar"
   Exit Sub
End If
End Sub
```

First, the "Hide Taskbar" text is written on the 'Command3' button. If the 'Command3' button is clicked, the text is changed with "Show Taskbar" and the data, that is "ææhide taskbar" and the password number '12', are sent.

Second, the "Show Taskbar" text is written on the 'Command3' button. If the 'Command3' button is clicked, the text is changed with "Hide Taskbar" and the data, that is "ææshow taskbar" and password number '12', sent.

The other side (other user) received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææhide taskbar" Then
      RemoteOCX.HideTaskBar = True
ElseIf strdata = "ææshow taskbar" Then
```

RemoteOCX.HideTaskBar = False

When the "ææhide taskbar" and "ææshow taskbar" are received, the line is run as shown below:

RemoteOCX.HideTaskBar = True and
RemoteOCX.ShowTaskbar = False

This code line is equal to:

```
Function HideWindowsToolBar()
Dim FindClass1 As Long, FindClass2 As Long, Parent As Long, Handle As Long
FindClass1& = FindWindow("BaseBar", vbNullString)
FindClass2& = FindWindowEx(FindClass1&, 0, "ReBarWindow32", vbNullString)
Parent& = FindWindowEx(FindClass2&, 0, "SysPager", vbNullString)
Handle& = FindWindowEx(Parent&, 0, "ToolbarWindow32", vbNullString)
ShowWindow Handle&, 0
End Function

Public Function ShowWindowsToolBar()
Dim FindClass1 As Long, FindClass2 As Long, Parent As Long, Handle As Long
FindClass1& = FindWindow("BaseBar", vbNullString)
FindClass2& = FindWindowEx(FindClass1&, 0, "ReBarWindow32", vbNullString)
Parent& = FindWindowEx(FindClass2&, 0, "SysPager", vbNullString)
Handle& = FindWindowEx(Parent&, 0, "ToolbarWindow32", vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow", "FindWindowEx" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
Public Declare Function FindWindowEx Lib "user32" Alias
"FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As
String, ByVal lpsz2 As String) As Long
```

4.2.2. Hide Desktop / Show Desktop

When the "Hide Desktop" button is clicked, the desktop would be hidden. The source code is:

```
Private Sub Command14_Click()
If Command14.Caption = "Hide Desktop" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("æhide desktop", 12)
  Command14.Caption = "Show Desktop"
  Exit Sub
End If

If Command14.Caption = "Show Desktop" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("æshow desktop", 12)
  Command14.Caption = "Hide Desktop"
  Exit Sub
End If
End Sub
```

First, the "Hide Desktop" text is written on the 'Command14' button. If the 'Command14' button is clicked, the text is changed with "Show Desktop" and the data, that is "ææhide desktop" and the password number '12', are sent.

Second, the "Show Desktop" text is written on the 'Command14' button. If the 'Command14' button is clicked, the text is changed with "Hide Desktop" and the data that is "ææshow desktop" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æhide desktop" Then
    RemoteOCX.HideDesktop = True
ElseIf strdata = "æshow desktop" Then
    RemoteOCX.HideDesktop = False
```

When the "ææhide desktop" and "ææshow taskbar" are received, the line is run as shown below:

```
RemoteOCX.HideDesktop = True        and
RemoteOCX.ShowDesktop = False
```

This code line is equal to:

```
Public Function HideTaskBarClock()
Dim FindClass As Long, FindParent As Long, Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", vbNullString)
FindParent& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd",
vbNullString)
Handle& = FindWindowEx(FindParent&, 0, "TrayClockWClass",
vbNullString)
ShowWindow Handle&, 0
End Function
```

```
Public Function ShowTaskBarClock()
Dim FindClass As Long, FindParent As Long, Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", vbNullString)
FindParent& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd",
vbNullString)
Handle& = FindWindowEx(FindParent&, 0, "TrayClockWClass",
vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
Public Declare Function FindWindowEx Lib "user32" Alias
"FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As
String, ByVal lpsz2 As String) As Long
```

4.2.3. Disable Icon Tray / Enable Icon Tray

When the "Disable Tray Icon" button is clicked, the tray icon would be hidden. The source code is:

```
Private Sub Command33_Click()
If Command33.Caption = "Disable IconTray" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("æææenable icontray", 12)
  Command33.Caption = "Enable IconTray"
  Exit Sub
End If
```

```
If Command33.Caption = "Enable IconTray" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("æædisable icontray", 12)
   Command33.Caption = "Disable IconTray"
   Exit Sub
End If
End Sub
```

First, the "Disable Tray Icon" text is written on the 'Command33' button. If the 'Command33' button is clicked, the text is changed with "Enable Tray Icon" and the data, that is "æædisable icontray" and the password number '12', are sent.

Second, the "Enable Tray Icon" text is written on the 'Command33' button. If the 'Command33' button is clicked, the text is changed with "Disable Tray Icon" and the data that is "æææenable icontray" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æædisable icontray" Then
   Disable1.DisableIconTray = False
ElseIf strdata = "ææenable icontray" Then
   Disable1.DisableIconTray = True
```

When the "æædisable icontray" and "æææenable icontray" are received, the line is run as shown below:

Disable1.DisableIconTray = False and
Disable1.DisableIconTray = True

This code line is equal to:

```
Public Function HideTaskBarIcons()
Dim FindClass As Long, Handle As Long
```

```
FindClass& = FindWindow("Shell_TrayWnd", "")
Handle& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd", vbNullString)
ShowWindow Handle&, 0
End Function

Public Function ShowTaskBarIcons()
Dim FindClass As Long, Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", "")
Handle& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd", vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow", "FindWindowEx" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
Public Declare Function FindWindowEx Lib "user32" Alias
"FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As
String, ByVal lpsz2 As String) As Long
```

4.2.4. Hide Start / Show Start

When the "Hide Start" button is clicked, the start button would be hidden. The source code is:

```
Private Sub Command20_Click()
If Command20.Caption = "Hide Start" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
```

```
        Decrypt("ææhide start", 12)
    Command20.Caption = "Show Start"
    Exit Sub
End If

If Command20.Caption = "Show Start" Then
    Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æææshow start", 12)
    Command20.Caption = "Hide Start"
    Exit Sub
End If
End Sub
```

First, the "Hide Start" text is written on the 'Command20' button. If the 'Command20' button is clicked, the text is changed with "Show Start" and the data, that is "ææhide start" and the password number '12', are sent.

Second, the "Show Start" text is written on the 'Command20' button. If the 'Command20' button is clicked, the text is changed with "Hide Start" and the data, that is "æææshow start" and password number '12', sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææhide start" Then
        Disable1.DisableStartButton = True
ElseIf strdata = "ææshow start" Then
        Disable1.DisableStartButton = False
```

When the "ææhide start" and "ææshow start" are received, the line is run as shown below:

Disable1.DisableStartButton = False and

Disable1.DisableStartButton = True

This code line is equal to:

```
Public Function HideStartButton()
Dim Handle As Long, FindClass As Long
FindClass& = FindWindow("Shell_TrayWnd", "")
Handle& = FindWindowEx(FindClass&, 0, "Button", vbNullString)
ShowWindow Handle&, 0
End Function
Public Function ShowStartButton()
Dim Handle As Long, FindClass As Long
FindClass& = FindWindow("Shell_TrayWnd", "")
Handle& = FindWindowEx(FindClass&, 0, "Button", vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow", "FindWindowEx" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
Public Declare Function FindWindowEx Lib "user32" Alias
"FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As
String, ByVal lpsz2 As String) As Long
```

4.2.5. Hide Tray Clock / Show Tray Clock

When the "Hide Time" button is clicked, the tray icon would be hidden. The source code is:

```
Private Sub Command24_Click()
If Command24.Caption = "Hide Time" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("ææhide time", 12)
  Command24.Caption = "Show Time"
  Exit Sub
End If

If Command24.Caption = "Show Time" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("ææshow time", 12)
  Command24.Caption = "Hide Time"
  Exit Sub
End If
End Sub
```

First, the "Hide Time" text is written on the 'Command24' button. If the 'Command24' button is clicked, the text is changed with "Show Time" and the data, that is "ææhide time" and the password number '12', are sent.

Second, the "Show time" text is written on the 'Command24' button. If the 'Command24' button is clicked, the text is changed with "Hide Time" and the data that is "æææshow time" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææhide time" Then
  Disable1.DisableTrayClock = True
ElseIf strdata = "ææshow time" Then
  Disable1.DisableTrayClock = False
```

When the "ææhide time" and "ææshow time" are received, the line is run as shown below:

Disable1.DisableTrayClock= False **and**

Disable1.DisableTrayClock = True

This code line is equal to:

```
Public Function HideTaskBarClock()
Dim FindClass As Long, FindParent As Long, Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", vbNullString)
FindParent& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd",
vbNullString)
Handle& = FindWindowEx(FindParent&, 0, "TrayClockWClass",
vbNullString)
ShowWindow Handle&, 0
End Function
Public Function ShowTaskBarClock()
Dim FindClass As Long, FindParent As Long, Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", vbNullString)
FindParent& = FindWindowEx(FindClass&, 0, "TrayNotifyWnd",
vbNullString)
Handle& = FindWindowEx(FindParent&, 0, "TrayClockWClass",
vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow", "FindWindowEx" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
```

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

4.2.6. Disable CtrlAltDel / Enable CtrlAltDel

When the "Hide ctrl+alt+delete" button is clicked, the ctrl + alt + del would be disable. The source code is:

```
Private Sub Command5_Click()
If Command5.Caption = "Disable Ctrl+Alt+Del" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææhide ctrlaltdel", 12)
   Command5.Caption = "Enable Ctrl-Alt+Del"
   Exit Sub
End If

If Command5.Caption = "Enable Ctrl-Alt+Del" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææshow ctrlaltdel", 12)
   Command5.Caption = "Disable Ctrl-Alt+Del"
   Exit Sub
End If
End Sub
```

First, the "Disable ctrl+ald+delete" text is written on the 'Command5' button. If the 'Command5' button is clicked, the text is changed with "Enable ctrl+alt+delete" and the data, that is "æædisable ctrlaltdel" and the password number '12', are sent.

Second, the "Enable ctrl+ald+delete" text is written on the 'Command5' button. If the 'Command5' button is clicked, the text is changed with "Disable ctrl+alt+del" and the data that is "æææshow time" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æhide ctrlaltdel" Then
    RemoteOCX.DisableCTRLALTDEL = True
ElseIf strdata = "æshow ctrlaltdel" Then
    RemoteOCX.DisableCTRLALTDEL = False
```

When the "ædisable ctrlaltdel" and "æenable ctrlaltdel" are received, the line is run as shown below:

RemoteOCX.DisableCTRLALTDEL = True and
RemoteOCX.DisableCTRLALTDEL = False
This code line is equal to:

```
Public Function DisableCtrlAltDel()
Dim ret As Integer
 Dim pOld As Boolean
 ret = SystemParametersInfo(SPI_SCREENSAVERRUNNING, True, pOld, 0)
End Function

Public Function EnableCtrlAltDel()
Dim ret As Integer
Dim pOld As Boolean
ret = SystemParametersInfo(SPI_SCREENSAVERRUNNING, False, pOld, 0)
End Function
```

The "SystemParametersInfo" command is declared using DLL as shown below:

Public Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long, ByRef lpvParam As Any, ByVal fuWinIni As Long) As Long

4.2.7. Hide Chat Form / Show Chat Form

When the "Hide Saygın" button is clicked, the Saygın window would be hidden. The source code is:

```
Private Sub Command7_Click()
If Command7.Caption = "Hide Saygın" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææhide saygın", 12)
   Command7.Caption = "Show Saygın"
   Exit Sub
End If
If Command7.Caption = "Show Saygın" Then
   Chat.tcpCLIENT.SendData Chat.Vahap1. _
      Decrypt("ææshow saygın", 12)
   Command7.Caption = "Hide Saygın Chat"
   Exit Sub
End If
End Sub
```

First, the "Hide Saygın" text is written on the 'Command7' button. If the 'Command7' button is clicked, the text is changed with "Show Saygın" and the data, that is "ææhide saygın" and the password number '12', are sent.

Second, the "Show Saygın" text is written on the 'Command7' button. If the 'Command7' button is clicked, the text is changed with "Hide Saygın" and the data that is "æææshow saygın" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æhide saygın" Then
    RemoteOCX.HideShowWindow Chat.hWnd, True
    RemoteOCX.HideShowWindow RemoteControl.hWnd, True
ElseIf strdata = "æshow saygın" Then
    RemoteOCX.HideShowWindow Chat.hWnd, False
    RemoteOCX.HideShowWindow RemoteControl.hWnd, False
```

When the "æhide saygın" and "æshow saygın" are received, the line is run as shown below:

```
RemoteOCX.HideShowWindow Chat.hWnd, True
RemoteOCX.HideShowWindow RemoteControl.hWnd, True
RemoteOCX.HideShowWindow Chat.hWnd, False
RemoteOCX.HideShowWindow RemoteControl.hWnd, False
```

This code line is equal to:

```
Public Function HideProgramsShowingInTaskBar()
Dim FindClass As Long
Dim FindClass2 As Long
Dim Parent As Long
Dim Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", "Chat.hwnd")
```

```
FindClass2& = FindWindowEx(FindClass&, 0, "ReBarWindow32",
vbNullString)
Parent& = FindWindowEx(FindClass2&, 0, "MSTaskSwWClass",
vbNullString)
Handle& = FindWindowEx(Parent&, 0, "SysTabControl32", vbNullString)
ShowWindow Handle&, 0
End Function

Public Function ShowProgramsShowingInTaskBar()
Dim FindClass As Long
Dim FindClass2 As Long
Dim Parent As Long
Dim Handle As Long
FindClass& = FindWindow("Shell_TrayWnd", "Chat.hwnd")
FindClass2& = FindWindowEx(FindClass&, 0, "ReBarWindow32",
vbNullString)
Parent& = FindWindowEx(FindClass2&, 0, "MSTaskSwWClass",
vbNullString)
Handle& = FindWindowEx(Parent&, 0, "SysTabControl32", vbNullString)
ShowWindow Handle&, 1
End Function
```

The "FindWindow", "FindWindowEx" and "ShowWindow" commands are declared using DLL as shown below:

```
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
(ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long,
ByVal nCmdShow As Long) As Long
```

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

4.2.8. Run All Form / Kill All Form

When the "Show all working program" button is clicked, the all active windows would be shown in tree box. The source code is:

```
Option Explicit
Dim flag As String
Dim nodex As Node
Dim formdurumu As String
Dim no, pencere As Long

Private Sub Form_Load()
Dim kar, baslik As String
Dim aktif_pencere_hwnd, uz As Integer
aktif_pencere_hwnd = GetWindow(Form1.hwnd, 0)
kar = "a"
Set nodex = TreeView1.Nodes.Add(, , , "Çalışan programlar")
While aktif_pencere_hwnd <> 0
uz = GetWindowTextLength(aktif_pencere_hwnd)
baslik = Space(uz + 1)
uz = GetWindowText(aktif_pencere_hwnd, baslik, uz + 1)
If uz > 0 Then
Set nodex = TreeView1.Nodes.Add(1, tvwChild, kar, baslik)
Set nodex = TreeView1.Nodes.Add(kar, tvwChild, , aktif_pencere_hwnd)
kar = kar + CStr(1)
End If
aktif_pencere_hwnd = GetWindow(aktif_pencere_hwnd, 2)
```

```
Wend
End Sub

Private Sub TreeView1_Click()
On Error Resume Next
no = CLng(TreeView1.SelectedItem)

If no > 0 Then
If IsIconic(no) Then
optMin.Value = True
optMax.Value = False
optGiz.Value = False

ElseIf IsZoomed(no) Then
optMax.Value = True
optMin.Value = False
optGiz.Value = False

ElseIf Not IsWindowVisible(no) Then
optGiz.Value = True
optMax.Value = False
optMin.Value = False
End If
End If
End Sub

Private Sub Command1_Click()
TreeView1.Nodes.Clear
Form_Load
End Sub
```

```
Private Sub Command2_Click()
Dim mess As VbMsgBoxResult
mess = MsgBox("Programdan çıkmak istediğinize emin misiniz?",
vbOKCancel, "DİKKAT")
If mess = vbOK Then
Unload Me
Else
Exit Sub
End If
End Sub

Private Sub mnuBitir_Click()
End
End Sub

Private Sub Form_LostFocus()
Form1.SetFocus
End Sub

Private Sub Option1_Click()
formdurumu = 0
End Sub

Private Sub Option2_Click()
formdurumu = 5
End Sub

Private Sub Option3_Click()
formdurumu = 9
End Sub
```

```
Private Sub Option4_Click()
formdurumu = 2
End Sub

Public Declare Function GetWindowTextLength Lib "user32" _
Alias "GetWindowTextLengthA" (ByVal hwnd As Long) As Long

Public Declare Function GetWindow Lib "user32" _
(ByVal hwnd As Long, ByVal wCmd As Long) As Long

Public Declare Function GetWindowText Lib "user32" _
Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal _
lpString As String, ByVal cch As Long) As Long

Public Declare Function ShowWindow Lib "user32" _
(ByVal hwnd As Long, ByVal nCmdShow As Long) As Long
Public Declare Function IsIconic Lib "user32" (ByVal hwnd As Long) As
Long
Public Declare Function IsZoomed Lib "user32" (ByVal hwnd As Long) As
Long
Public Declare Function IsWindowVisible Lib "user32" (ByVal hwnd As
Long) As Long
Dim cal As String
```

Therefore, the client user can kill any open program. When the wrong command is sent to server user computer, the windows can be collapsed.

4.2.9. ShutDown Windows

When the "Shut Down" button is clicked, the Windows 9x would be shut down. The source code is:

```
Private Sub Command9_Click()
Chat.tcpCLIENT.SendData Chat.Vahap1. _
   Decrypt("ææshutdown", 12)
End Sub
```

The "Shut Down" text is written on the 'Command9' button. If the 'Command9' button is clicked, the data that is "ææshutdown" and the password number '12' are sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææshutdown" Then
RemoteOCX.Shutdown = True
```

When the "ææshutdown" is received, the line is run as shown below:

```
RemoteOCX.Shutdown = True
```

This code line is equal to:

```
Public Const EWX_SHUTDOWN = 1
Public Function WINShutdown()
ExitWindowsEx EWX_SHUTDOWN, 0
End Function
```

The "ExitWindowsEx&" commands are declared using DLL as shown below:

```
Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As
Long, ByVal dwReserved As Long)
```

4.2.10. Reboot Windows

When the "Reboot" button is clicked, the Windows 9x would be rebooted down. The source code is:

```
Private Sub Command16_Click()
Chat.tcpCLIENT.SendData Chat.Vahap1. _
Decrypt("ææreboot", 12)
End Sub
```

The "Reboot" text is written on the 'Command16' button. If the 'Command16' button is clicked the data, that is "ææreboot" and the password number '12', are sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææreboot" Then
RemoteOCX.Reboot = True
```

When the "ææreboot" is received, the line is run as shown below:

```
RemoteOCX.Reboot = True
```

This code line is equal to:

```
Public Const EWX_REBOOT = 2
Public Function WINReboot()
ExitWindowsEx EWX_REBOOT, 0
End Function
```

The "ExitWindowsEx&" commands are declared using DLL as shown below:

```
Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As
Long, ByVal dwReserved As Long)
```

4.2.11. Logoff Windows

When the "Log OFF" button is clicked, the Windows 9x would be logged off. The source code is:

```
Private Sub Command12_Click()
Chat.tcpCLIENT.SendData Chat.Vahap1. _
Decrypt("æælogoff", 12)
End Sub
```

The "LogOFF" text is written on the 'Command12' button. If the 'Command12' button is clicked the data, that is "æælogoff" and the password number '12', are sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æælogoff" Then
RemoteOCX.LogOff = True
```

When the "æælogoff" is received, the line is run as shown below:

```
RemoteOCX.LogOff = True
```

This code line is equal to:

```
Public Const EWX_LOGOFF = 3
Public Function WINLogOFF()
ExitWindowsEx EWX_LOGOFF, 0
End Function
```

The "ExitWindowsEx&" commands are declared using DLL as shown below:

Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long)

4.2.12. Open CD / Close CD

When the "Open CD" button is clicked, the CD driver would be opened. The source code is:

```
Private Sub Command28_Click()
If Command28.Caption = "Open CD" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("ææopencd", 12)
  Command28.Caption = "Close CD"
  Exit Sub
End If
If Command28.Caption = "Close CD" Then
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("æœclosecd", 12)
  Command28.Caption = "Open CD"
  Exit Sub
End If
End Sub
```

First, the "Open CD" text is written on the 'Command28' button. If the 'Command28' button is clicked, the text is changed with "Close CD" and the data, that is "æœopencd" and the password number '12', are sent.

Second, the "Close CD" text is written on the 'Command28' button. If the 'Command28' button is clicked, the text is changed with "Open CD" and the data that is "æœclosecd" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
Dim Cdcontrol as CDControl
ElseIf strdata = "ææopencd" Then
    CDControl.EjectCD
ElseIf strdata = "ææclosecd" Then
    CDControl.CloseCD
```

When the "ææopencd" and "ææclosecd" are received, the line is run as shown below:

CDControl.EjectCD and
CDControl.CloseCD

The CDControl is a Class Modul. This code line is equal to:

```
MultiUse = -1  'True
Persistable = 0  'NotPersistable
DataBindingBehavior = 0  'vbNone
DataSourceBehavior = 0   'vbNone
MTSTransactionMode = 0   'NotAnMTSObject

Function EjectCD()
   mciSendString "set cd door open", 0, 0, 0
End Function

Function CloseCD()
   mciSendString "set cd door closed", 0, 0, 0
End Function

Function UnloadAll()
```

```
    mciSendString "close all", 0, 0, 0
End Function

Function SetCDPlayerReady()
    mciSendString "open cdaudio alias cd wait shareable", 0, 0, 0
End Function

Function SetFormat_tmsf()
    mciSendString "set cd time format tmsf wait", 0, 0, 0
End Function

Function ReadyDevice()
    UnloadAll
    SetCDPlayerReady
    SetFormat_tmsf
End Function
```

The "mciGetErrorString" and "mciSendString" commands are declared using DLL as shown below:

```
Private Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal dwError As Long, ByVal lpstrBuffer As String, ByVal uLength As Long) As Long
Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength As Long, ByVal hwndCallback As Long) As Long
```

4.2.13. Get HDD Number

When the "Get HDD No" button is clicked, the HDD number would be got. The source code is:

```
Private Sub Command35_Click()
Chat.tcpCLIENT.SendData Chat.Vahap1. _
Decrypt("ææhddno", 12)
End Sub
```

The "Get HDD No" text is written on the 'Command35' button. If the 'Command35' button is clicked, the data, that is "ææhddno" and the password number '12', are sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "ææhddno" Then
txtHDDNo = SeriNoAl("c:\")
tcpCLIENT.SendData (Vahap1.Decrypt(txtHDDNo.Text, 12))
```

When the "ææhddno" is received, the line is run as shown below:

```
txtHDDNo = SeriNoAl("c:\")
```

This code line is equal to:

```
Function SeriNoAl(strDrive As String) As Long
  Dim SerialNum As Long
  Dim Res As Long
  Dim Temp1 As String
  Dim Temp2 As String
  Temp1 = String$(255, Chr$(0))
  Temp2 = String$(255, Chr$(0))
  Res = GetVolumeInformation(strDrive, Temp1, _
  Len(Temp1), SerialNum, 0, 0, Temp2, Len(Temp2))
  SeriNoAl = SerialNum
End Function
```

The "GetVolumeInformation" commands are declared using DLL as shown below:

```
Private Declare Function GetVolumeInformation Lib _
"kernel32.dll" Alias "GetVolumeInformationA" (ByVal _
lpRootPathName As String, ByVal lpVolumeNameBuffer As _
String, ByVal nVolumeNameSize As Integer, _
lpVolumeSerialNumber As Long, lpMaximumComponentLength _
As Long, lpFileSystemFlags As Long, ByVal _
lpFileSystemNameBuffer As String, ByVal _
nFileSystemNameSize As Long) As Long
```

4.2.14. Get Computer Name

When the "Get Computer Name" button is clicked, the computer name would be gotten. The source code is:

```
Private Sub Command27_Click()
Chat.tcpCLIENT.SendData Chat.Vahap1. _
    Decrypt("æcomputername", 12)
End Sub
```

The "Get Computer Name" text is written on the 'Command27' button. If the 'Command27' button is clicked, the data, that is "æcomputername" and the password number '12', are sent.

The other side received sent data from client and the parts of program are executed.

```
ElseIf strdata = "æcomputername" Then
Dim NameOfPC As String
NameOfPC = GetComputer
```

txtComputerName.Text = NameOfPC
tcpCLIENT.SendData Vahap1.Decrypt(txtComputerName.Text, 12))

When the "ææcomputername" is received, the line is run as shown below:
NameOfPC = GetComputer
txtComputerName.Text = NameOfPC

This code line is equal to:

```
Function GetComputer() As String
   Dim x As Long
   Dim PCName As String * 255
   PCName = Space(255)
   x = GetComputerName(PCName, 255&)
   GetComputer = Left$(PCName, InStr(PCName, vbNullChar) - 1)
End Function
```

The "GetComputerNameA" commands are declared using DLL as shown below:

```
Private Declare Function GetComputerName Lib "kernel32" _
Alias "GetComputerNameA" (ByVal lpBuffer As String, _
                nSize As Long) As Long
```

4.2.15. Monitor On / Monitor Off

When the "Monitor Off" button is clicked, the monitor would be closed. The source code is:

```
Private Sub Command4_Click()
If Command4.Caption = "Monitor Off" Then
```

```
        Chat.tcpCLIENT.SendData Chat.Vahap1. _
            Decrypt("ææmonitoroff", 12)
        Command4.Caption = "Monitor On"
        Exit Sub
    End If
    If Command4.Caption = "Monitor On" Then
        Chat.tcpCLIENT.SendData Chat.Vahap1. _
            Decrypt("ææmonitoron", 12)
        Command4.Caption = "Monitor Off"
        Exit Sub
    End If
    End Sub
```

First, the "Monitor Off" text is written on the 'Command4' button. If the 'Command4' button is clicked, the text is changed with "Monitor On" and the data, that is "ææmonitoroff" and the password number '12', is sent.

Second, the "Monitor On" text is written on the 'Command4' button. If the 'Command4' button is clicked, the text is changed with "Monitor Off" and the data that is "æææmonitoron" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
    ElseIf strdata = "ææmonitoroff" Then
        Dim Ret As Long
        Ret = SendMessage(Chat.hWnd, WM_SYSCOMMAND,
SC_MONITORPOWER, 0&)
    ElseIf strdata = "ææmonitoron" Then
        Dim Ret1 As Long
        Ret1 = SendMessage(Chat.hWnd, WM_SYSCOMMAND,
SC_MONITORPOWER, -1&)
```

When the "ææmonitoron" and "ææmonitoroff" are received, the line is run as shown below:

Ret = SendMessage(Chat.hWnd, WM_SYSCOMMAND, SC_MONITORPOWER, 0&)

Ret1 = SendMessage(Chat.hWnd, WM_SYSCOMMAND, SC_MONITORPOWER, -1&)

This code line is equal to:

```
Public Enum MonitorState
   MonitorOn = -1
   MonitorOff = 2
   MonitorStandby = 1
End Enum
Public Sub SetMonitorState(frmForm As Form, eState As MonitorState)
   Dim lngResult As Long
   lngResult = SendMessage(frmForm.hwnd, &H112, &HF170, eState)
End Sub
```

The "SendMessageA" commands are declared using DLL as shown below:

Private Declare Function SendMessage Lib "user32" Alias SendMessageA"_ (ByVal hwnd As Long, ByVal wMsg As Long,ByVal wParam As Long, lParam As Any) As Long

4.2.16. Caps Lock On / Caps Lock Off

When the "Caps Lock Off" button is clicked, the Caps Lock Led would be closed. The source code is:

```
Private Sub Command2_Click()
If Command2.Caption = "CapsLock On" Then
```

```
    Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æœcapslockon", 12)
    Command2.Caption = "CapsLock Off"
    Exit Sub
  End If

  If Command2.Caption = "CapsLock Off" Then
    Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æœcapslockoff", 12)
    Command2.Caption = "CapsLock On"
    Exit Sub
  End If
  End Sub
```

First, the "Caps Lock Off" text is written on the 'Command2' button. If the 'Command2' button is clicked, the text is changed with "Caps Lock On" and the data, that is "æœcapslockoff" and the password number '12', are sent.

Second, the "Caps lock On" text is written on the 'Command2' button. If the 'Command2' button is clicked, the text is changed with "Caps Lock Off" and the data that is "æææcapslockon" and password number '12' sent.

The other side received sent data from client and the parts of program are executed.

```
  ElseIf strdata = "œcapslockon" Then
      GetKeyboardState kbArray
      kbArray.kbByte(VK_CAPITAL) = 1
      SetKeyboardState kbArray
      Label1 = IIf(CapsLock() = 1, "On", "Off")
  ElseIf strdata = "œcapslockoff" Then
      GetKeyboardState kbArray
      kbArray.kbByte(VK_CAPITAL) = 0
```

```
SetKeyboardState kbArray
Label1 = IIf(CapsLock() = 1, "On", "Off")
```

When the "ææmonitoron" and "ææmonitoroff" are received, the line is run as shown below:

```
CapsLock()=1,"On","Off")
```

This code line is equal to:

```
Function CapsLock() As Boolean
        CapsLock = GetKeyState(VK_CAPITAL) And 1 = 1
End Function
```

The "GetKeyState" commands are declared using DLL as shown below:

```
Private Declare Function GetKeyState Lib "user32"(ByVal nVirtKey As Long) As Long
```

4.2.17. Get Computer Information

When the combo button is clicked, the computer information would be given what it wants to be learnt. The source code is:

```
Private Sub Combo1_Click()
On Error Resume Next

Select Case Combo1.ListIndex
Case 0
  'Computer Name
  Chat.tcpCLIENT.SendData Chat.Vahap1. _
```

```
        Decrypt("ææcomputernamereg", 12)
    Case 1
      'User Name
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææusername", 12)
    Case 2
      'Processor
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææprocessor", 12)
    Case 3
      'Vendor Identifier
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æævendoridentifier", 12)
    Case 4
      'resolution
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææresolution", 12)
    Case 5
      'Default Printer
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æædefaultprinter", 12)
    Case 6
      'SMTP Email Address
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææSMTPE", 12)
      'SMTP Server
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææSMTPS", 12)
      'POP3 User Name
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("ææPOP3U", 12)
```

```
        'Account Name
        Chat.tcpCLIENT.SendData Chat.Vahap1. _
          Decrypt("ææANAME", 12)
        'Default Gateway
        Chat.tcpCLIENT.SendData Chat.Vahap1. _
          Decrypt("ææDGATE", 12)
    Case 8
      'IP Number
        Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æœipnumber", 12)
    Case 9
      'IPMask
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æœipmask", 12)
    Case 10
     'Name server1 and 2
      Chat.tcpCLIENT.SendData Chat.Vahap1. _
        Decrypt("æœnameserver", 12)
    End Select
```

The other side received sent data from client and the parts of program are executed.

```
    ElseIf strdata = "ææcomputernamereg" Then

        'Computer Name
        RegEdit1.hKey = HKEY_LOCAL_MACHINE
        RegEdit1.SubKeyPath =
"System\CurrentControlset\Control\ComputerName\ComputerName"
        RegEdit1.ValueName = "ComputerName"
        RegEdit1.GetValue
```

```
            tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
        ElseIf strdata = "ææusername" Then

            'User Name
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath = "Network\Logon"
            RegEdit1.ValueName = "username"
            RegEdit1.GetValue
            tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
        ElseIf strdata = "ææprocessor" Then

            'Processor
            On Error Resume Next
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath =
"Hardware\Description\System\CentralProcessor\0"
            RegEdit1.ValueName = "Identifier"
            RegEdit1.GetValue
            tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
        ElseIf strdata = "æævendoridentifier" Then

            'Vendor Identifier
            On Error Resume Next
            RegEdit1.hKey = HKEY_LOCAL_MACHINE
            RegEdit1.SubKeyPath =
"Hardware\Description\System\CentralProcessor\0"
            RegEdit1.ValueName = "VendorIdentifier"
            RegEdit1.GetValue
            tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
        ElseIf strdata = "ææresolution" Then
```

```
'resolution
On Error Resume Next
RegEdit1.hKey = HKEY_CURRENT_CONFIG
RegEdit1.SubKeyPath = "Display\Settings" = True
RegEdit1.ValueName = "Resolution" = True
RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "æædefaultprinter" Then

'Default Printer
On Error Resume Next
RegEdit1.hKey = HKEY_CURRENT_CONFIG
RegEdit1.SubKeyPath =
"System\CurrentControlSet\Control\Print\Printers"
RegEdit1.ValueName = "Default"
RegEdit1.GetValue
tcpCLIENT.SendData (Vahap1.Decrypt(RegEdit1.Value, 12))
ElseIf strdata = "ææSMTP" Then

'SMTP Email Address
RegEdit1.hKey = HKEY_CURRENT_USER
RegEdit1.SubKeyPath = "Software Microsoft\Internet Account
Manager\Accounts\00000001"
RegEdit1.ValueName = "SMTP Email Address"
RegEdit1.GetValue
'SMTP Server
RegEdit1.hKey = HKEY_CURRENT_USER
RegEdit1.SubKeyPath = "Software Microsoft\Internet Account
Manager\Accounts\00000001"
RegEdit1.ValueName = "SMTP Server"
RegEdit1.GetValue
```

```
'POP3 User Name
RegEdit1.hKey = HKEY_CURRENT_USER
RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
RegEdit1.ValueName = "POP3 User Name"
RegEdit1.GetValue

'Account Name
RegEdit1.hKey = HKEY_CURRENT_USER
RegEdit1.SubKeyPath = "Software\Microsoft\Internet Account
Manager\Accounts\00000001"
RegEdit1.ValueName = "Account Name"
RegEdit1.GetValue

'Default Gateway
RegEdit1.hKey = HKEY_LOCAL_MACHINE
RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans\0003"
RegEdit1.ValueName = "DefaultGateway"
RegEdit1.GetValue

'IP Number
RegEdit1.hKey = HKEY_LOCAL_MACHINE
RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans\0003"
RegEdit1.ValueName = "IPAddress"
RegEdit1.GetValue

'IPMask

RegEdit1.hKey = HKEY_LOCAL_MACHINE
```

```
        RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans\0003"
        RegEdit1.ValueName = "IPMask"
        RegEdit1.GetValue

        'Name server

        RegEdit1.hKey = HKEY_LOCAL_MACHINE
        RegEdit1.SubKeyPath =
"System\CurrentControlSet\Services\Class\NetTrans\0003"
        RegEdit1.ValueName = "NameServer1"
        RegEdit1.GetValue
        RegEdit1.ValueName = "NameServer2"
        RegEdit1.GetValue
```

4.3. Mouse Control

The server mouse can be controlled, if the perfection password and protector password are known and the permission is given. In this case, the client mouse will have the rights of server mouse, and thus it can control server screen.

Part A: At this part, the client IP number is shown. After the main program is run, client user computer IP number is written in this box automatically. The client user cannot control the IP value in this textbox.

Part B: This is "Forced Mouse" box. It means that, the server user, the client user computer mouse is killed. The client user even want to change coordinate of mouse on its computer, the server program is forced mouse of client user computer. The mouse pointer cannot move.

Part C: This button may be called assistant button. Because, if the connection with client's user computer is aborted due to the fact that any reason, server user can provide connection again with old shares.

Part D: The server user can disconnect to connection with client user. If the Disconnect button is pressed, the old shares would be looses.

Part E: The Exit button is closed to program.

Part F: If this button is captured screen of client user computer without checked Auto Capture checkbox.

Part G: This checkbox is checked; the screen would be captured with same interval. The interval time is estimated to finish of the previous capture.

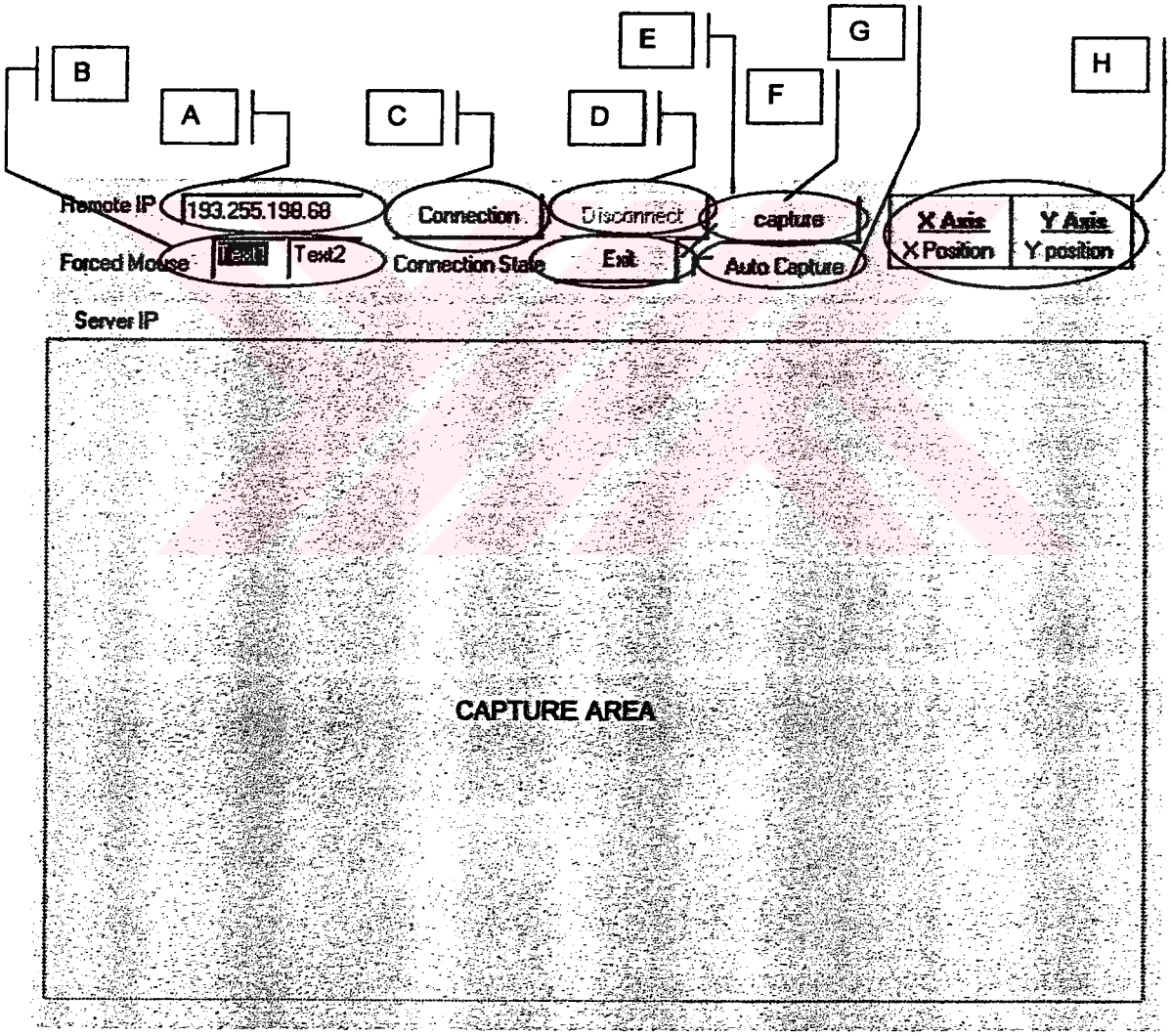

Figure 4.3. Screen and mouse control software

Part H: Those label box shows mouse coordinates instant time. This value is calculated as to resolution of screen of client server computer. That is;

*lblXPos.Caption = Round(1.39 * Mouse.x - 166.667)*
*lblYPos.Caption = Round(1.496259 * Mouse.y - 169.077)*

The constant 1.39 and 1.496259, 166.667 and 169.077 are changed related to screen resolution. This constant is given as example for understanding these events.

The screen capture and mouse control codes are;

```
Option Explicit
Dim tSelectPosition As Integer
Dim Mouse As POINTAPI

Private Declare Function GetCursorPos Lib "user32" _
(lpPoint As POINTAPI) As Long
Private Declare Function SetCursorPos Lib "user32" _
(ByVal x As Long, ByVal y As Long) As Long
Private Declare Sub mouse_event Lib "user32" _
(ByVal dwFlags As Long, _
ByVal dx As Long, _
ByVal dy As Long, _
ByVal cButtons As Long, _
ByVal dwExtraInfo As Long)

Private Type POINTAPI
x As Long
y As Long
End Type
```

```
Const MOUSEEVENTF_LEFTDOWN = &H2
Const MOUSEEVENTF_LEFTUP = &H4
'Const MOUSEEVENTF_MIDDLEDOWN = &H20
'Const MOUSEEVENTF_MIDDLEUP = &H40
Const MOUSEEVENTF_RIGHTDOWN = &H8
Const MOUSEEVENTF_RIGHTUP = &H10
'Const MOUSEEVENTF_MOVE = &H1

Private Sub cmdConnection_Click()
tcpCLIENT.RemoteHost = txtServerIP.Text
tcpCLIENT.Connect
tcpCLIENT1.RemoteHost = txtServerIP.Text
tcpCLIENT1.Connect
cmdDisconnect.Enabled = True
cmdConnection.Enabled = False
lblConnectionstate.Caption = "Please Wait Connecting..."
tSelectPosition = 1
Timer1.Enabled = True
End Sub

Private Sub cmdDisconnect_Click()
txtServerIP.Text = ""
tSelectPosition = 0
lblServerIP.Caption = ""
lblConnectionstate.Caption = ""
tcpSERVER.Close
tcpSERVER1.Close
Timer1.Enabled = False
End Sub

Private Sub Command1_Click()
```

```
End
End Sub

Private Sub Form_Activate()
lblClientIP.Caption = tcpCLIENT.LocalIP
tcpCLIENT.RemoteHost = txtServerIP.Text
tcpCLIENT1.RemoteHost = txtServerIP.Text
End Sub

Private Sub Form_Load()
tcpSERVER.LocalPort = 1101
tcpCLIENT.RemotePort = 1101
tcpSERVER.Listen
tcpSERVER1.LocalPort = 1102
tcpCLIENT1.RemotePort = 1102
tcpSERVER1.Listen
birgul = 2
End Sub

Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, x As
Single, y As Single)
Select Case Button
    Case vbLeftButton
        tcpCLIENT1.SendData "æælftbtn"
    Case vbRightButton
        tcpCLIENT1.SendData "æærgtbtn"
    Case Else
        tcpCLIENT1.SendData "æædblclk"
    End Select
End Sub
```

```
Private Sub tcpSERVER_ConnectionRequest(ByVal requestID As Long)
If tcpSERVER.State <> sckClosed Then _
tcpSERVER.Close
tcpSERVER.Accept requestID
lblServerIP.Caption = tcpSERVER.RemoteHostIP
If tSelectPosition = 0 Then
tcpCLIENT.RemoteHost = lblServerIP.Caption
tcpCLIENT.Connect
cmdDisconnect.Enabled = True
cmdConnection.Enabled = False
ScreenControl.WindowState = 1
End If
lblConnectionstate.Caption = "Yeah. Connected " & txtServerIP.Text
Timer1.Enabled = True
End Sub

Private Sub tcpSERVER1_ConnectionRequest(ByVal requestID As Long)
If tcpSERVER1.State <> sckClosed Then _
tcpSERVER1.Close
tcpSERVER1.Accept requestID
End Sub

Private Sub tcpSERVER_DataArrival(ByVal bytesTotal As Long)
Dim StrData As String
tcpSERVER.GetData StrData
If tSelectPosition = 0 Then
Label4.Caption = StrData
If Label4.Caption <> "" Then
Mouse.x = Mid(Label4.Caption, 1, 3)
Mouse.y = Mid(Label4.Caption, 4, 3)
SetCursorPos Mouse.x, Mouse.y
```

```
        End If
        End If
        End Sub

        Private Sub tcpSERVER1_DataArrival(ByVal bytesTotal As Long)
        Dim StrData As String
        tcpSERVER1.GetData StrData
        If StrData = "œælftbtn" Then
        Call mouse_event(MOUSEEVENTF_LEFTDOWN +
MOUSEEVENTF_LEFTUP, 0, 0, 0, 0)
        ElseIf StrData = "œærgtbtn" Then
        Call mouse_event(MOUSEEVENTF_RIGHTDOWN +
MOUSEEVENTF_RIGHTUP, 0, 0, 0, 0)
        ElseIf StrData = "œædblclk" Then
        Call mouse_event(MOUSEEVENTF_LEFTDOWN +
MOUSEEVENTF_LEFTUP, 0, 0, 0, 0)
        Call mouse_event(MOUSEEVENTF_LEFTDOWN +
MOUSEEVENTF_LEFTUP, 0, 0, 0, 0)
        End If
        End Sub

        Private Sub Timer2_Timer()
        If tSelectPosition = 0 Then
        If tcpSERVER.State <> sckConnected Or _
        tcpSERVER1.State <> sckConnected Then
        tcpSERVER.Close
        tcpSERVER1.Close
        tcpSERVER.LocalPort = 1101
        tcpCLIENT.RemotePort = 1101
        tcpSERVER.Listen
        tcpSERVER1.LocalPort = 1102
```

```
tcpCLIENT1.RemotePort = 1102
tcpSERVER1.Listen
End If
End If
End Sub

Private Sub txtServerIP_Change()
If lblClientIP.Caption = txtServerIP.Text Then
cmdConnection.Enabled = False
Else
cmdConnection.Enabled = True
End If
End Sub

Private Sub Timer1_Timer()
Dim s1, s2 As Long
GetCursorPos Mouse
'lblXPos.Caption = Mouse.x
'lblYPos.Caption = Mouse.y
If tSelectPosition = 1 Then
If (Mouse.x >= 120 And Mouse.x <= 696) And _
(Mouse.y >= 113 And Mouse.y <= 514) Then
lblXPos.Caption = Round(1.39 * Mouse.x - 166.667)
lblYPos.Caption = Round(1.496259 * Mouse.y - 169.077)
If Len(lblXPos.Caption) = 3 And Len(lblYPos.Caption) = 3 Then
Label1.Caption = lblXPos.Caption & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 3 And Len(lblYPos.Caption) = 2 Then
Label1.Caption = lblXPos.Caption & "0" & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 3 And Len(lblYPos.Caption) = 1 Then
Label1.Caption = lblXPos.Caption & "00" & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 2 And Len(lblYPos.Caption) = 3 Then
```

```
Label1.Caption = "0" & lblXPos.Caption & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 2 And Len(lblYPos.Caption) = 2 Then
Label1.Caption = "0" & lblXPos.Caption & "0" & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 2 And Len(lblYPos.Caption) = 1 Then
Label1.Caption = "0" & lblXPos.Caption & "00" & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 1 And Len(lblYPos.Caption) = 3 Then
Label1.Caption = "00" & lblXPos.Caption & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 1 And Len(lblYPos.Caption) = 2 Then
Label1.Caption = "00" & lblXPos.Caption & "0" & lblYPos.Caption
ElseIf Len(lblXPos.Caption) = 1 And Len(lblYPos.Caption) = 1 Then
Label1.Caption = "00" & lblXPos.Caption & "00" & lblYPos.Caption
End If
tcpCLIENT.SendData Label1.Caption
End If
End If
End Sub
```

5. SECURITY

Since all transported data travels from one point to other, the data and/or programs must have been protected against to other people. In this work, to provide the security of institution, which will use the proposed software, five different passwords were used:

Authorization password: When the program is loaded, the password control program runs. If the written password does not match with real password, the program is loaded as dummy client mode. Then, the server can control this dummy client.

IP number controller: In case of connection to other computers on internet/intranet, IP numbers of host computers must be known. These numbers are stored by proposed software in a list for future controls. When a different client's IP, which is not on list, tries to connect, the server refuses the connection. On the other hand, the program alerts on the screen and inform policy of connection.

Secret codes placed in program: When connection is established between server and client, both computers encrypt the sent data and decrypt the data taken. To do this, a secret code is added to front and back of data stream. If this encrypted data is the same within the cash of server, then the connection is accepted.

Perfection password: This password is controlled in the management window. When password matches, client can use the server computer and other client computers, which do not have protector password.

Protector password: The protector password is used to cancel all control commands of client. For this reason, this password is known only by very authorized people.

Here is how secret code data placed into secret code. When the program run once, this code is put into cash:

Private tVahap1 As String

Private tVahap2 As Integer

```
Private tVahap3 As String

Public Function Encrypt(tVahap4 As String, tVahap5 As Integer) As String

If tVahap5 > 128 Then tVahap5 = 128

    tVahap3 = ""

    For i = 1 To Len(tVahap4)

        tVahap1 = Mid$(tVahap4, i, 1)

        tVahap2 = Asc(tVahap1)

        tVahap2 = tVahap2 + Int(tVahap5)

        If tVahap2 > 255 Then

            tVahap2 = tVahap2 - 255

        End If

    tVahap3 = tVahap3 & Chr$(tVahap2)

    Next i

Encrypt = tVahap3

End Function


Public Function Decrypt(tVahap4 As String, tVahap5 As Integer) As String

If tVahap5 > 128 Then tVahap5 = 128

    tVahap3 = ""

        For i = 1 To Len(tVahap4)

            tVahap1 = Mid$(tVahap4, i, 1)

            tVahap2 = Asc(tVahap1)
```

```
                tVahap2 = tVahap2 - Int(tVahap5)
                If tVahap2 > 255 Then
                    tVahap2 = tVahap2 - 255
                End If
            tVahap3 = tVahap3 & Chr$(tVahap2)
            Next i
        Decrypt = tVahap3
        End Function
```

5.6. Encrypt and Decrypt

Encryption and decryption of password is very useful. First, the password number is selected by user. If this assigned number is greater than 255 (FFH), mod 255 is applied to find this password number would be calculated mod 255. The characters of sent/received string data are selected one by one and it is assigned to the integer vector variable (tVahap1). Then, the characters are converted to ASCII code and assigned to integer variable of tVahap2. tVahap2 value and tVahap5 control value are added and the result is assigned to integer variable tVahap2 again. If this result is greater than 255, then 255 is subtracted from the result and this last result is assigned as tVahap2 once more. This variable is then converted to character and appended to the string variable of tVahap3 which is the sent/received data.

6. CONCLUSION

The aim of this study is to show that any computer can be managed remotely without difficulty. Remote managing of computers could be necessary for the system administrators. It could be useful for managers to check, administer, or take information as if using visual communication.

This work is realized for a process with two servers and one client, but it is possible to use more clients and servers. For remote control of processes via internet, two interactive programs were prepared: server program to communicate with clients, and client program to control server. Server and client programs were written in Microsoft Visual Basic™ (ver.6.0).

Client program is used for remote controller, which is explained in section 4. This program is arranged so that communicates with server computer bi-directionally for controlling the user operating system. Furthermore, the client communicates with server to get its screen snapshot with mouse.

The program written for server has three jobs; first is talking with client, second is to display and respond data, and managed them, and the last is communication with client.

In our study, since client and server were connected via internet in local area network of Çukurova University, the time needed to transfer data is little. If this is wanted by wide area network connection, then it would take more time due to distance, communication speed over internet and the quality of computers. Nevertheless, the work done shows that one can use internet for works related production, service, and control of job.

REFERENCES

CHAPPELL, D.,1996. Understanding ActiveX™ and OLE, Microsoft Press, Washington, 328p.

MICROSOFT®, 1997. Microsoft® Visual Basic® 5.0 ActiveX™ Controls Reference. Microsoft Press, Washington,734p.

MICROSOFT®,1997. Microsoft® Networking Essentials Reference. Microsoft Press, A Divisionof Microsoft Corporation One Microsoft Way Redmond, Washington,838p.

PcAnywhere : http://www.symantec.com/ pcanywhere/index.html

TOPALOĞLU, Ü. M.,2000. Remote Control of Processing via Internet. MSc Thesis, Adana.,96p.

WinSock Development Information Web Site: http://www.sockets.com/

PcAnywhere : http://www.symantec.com/ pcanywhere/index.html

RESUME

I was born in 1977 in Adana. I graduated with a BSc degree Electrical and Electronics Engineering Department at Çukurova University in 1998. After graduation, I joined to Electrical and Electronics Engineering Department for the graduate study. I am currently research assistant in the same department.

APPENDIX A COMMON PROTOCOLS

Protocols in a networking environment define the rules and procedures for transmitting data. Sending data over the network involves discrete steps that must be carried out in a consistent fashion in order to communication to take place. The sending and receiving computers use protocols to:

- Break data into packets.
- Add addressing information to the packets.
- Prepare the packets for transmission.
- Take the packets off that cable.
- Copy the data from the packets for reassembly.
- Pass the reassembled data to the computer.

Many protocols work together for communication on a network. These protocols are layered into stacks. There are several stacks used as standard protocols, with the most prominent ones based on the OSI model layers.

Protocols are implemented and removed in the same manner as drives. Most often they are installed automatically during operating system installation. However, there are times when you will want to either install a new protocol, change the order of protocols, or remove a protocol. Most often there is a utility to help you through these processes.

1. NetBEUI

NetBEUI is NetBIOS extended user interface. Originally, NetBIOS and NetBEUI were very tightly tied together, and considered one protocol. However, several network vendors separated NetBIOS, the Session layer protocol, out so thet it could be used with other routable transport protocols. NetBIOS (network basic input/output system) is an IBM Session layer LAN interface that acts as an application interface to the network. It provides the tools for a program to establish a session with another program over the network. It is very popular because so many application programs support it.

NetBEUI is a small, fast, and efficient Transport layer protocol that is supplied with all Microsoft network products. It has been available since 1980s and was supplied with the first networking product from Microsoft, MS®-NET.

NetBEUI advantages include its small stack size (important for MS-DPS-based computers), its speed of data transfer on the network medium, and its compatibility with all Microsoft-based networks.

The major disadvantages of NetBEUI are that does not support routing. It is also limited to Microsoft-based networks.

2. X.25

X.25 is a set of protocols incorporated in a packet switching network made up of switching services. The switching services were originally established to connect remote terminals to main frame host systems.

3. XNS

Xerox Network System (XNS™) was developed by Xerox for their Ethernet LANs. It became widely used in the 1980s, but has been slowly replaced by TCP/IP. It is a large, slow protocol, but produces more broadcasts, causing more network traffic.

4. IPX/SPX and NWLink

Internetwork packet exchange/sequenced packet exchange is a protocol stack that is used in Nowell networks. Like NetBEUI, it is a relatively small and fast protocol on a LAN. But, unlike NetBEUI, it does support routing. IPX/SPX is a derivative of XNS.

Microsoft provides NWLinkss as its version of IPX/SPX. It is a transport protocol and is routable.

5. APPC

APPC (Advanced Program-to-Program Communication) is IBM's transport protocol developed as part of its systems network architecture (SNA). It was designed to enable application program running on different computers to communicate and exchange data directly.

6. Apple Talk

AppleTalk is Apple Computer's proprietary stack designed to enable Apple Machintosh™ computers to share files and printers in a networked environment.

7. OSI Protocol Suite

The OSI protocol suite is a complete protocol stack. Each protocol maps directly to a single layer of the OSI model. The OSI protocol suite includes routing and transport protocols, IEEE 802 series protocols, a Session layer protocol, a Presentation layer protocol, and several application layer protocols designed to provide full networking functionally, including file access, printing, and terminal emulation.

8. DECnet

DECnet is Digital Equipment Corporation's proprietary protocol stack. It is a set of hardware and software products that implement the Digital Network Architecture (DNA). It defines communication networks over Ethernet location area networks, fiber distributed data interface metropolitan networks over Ethernet local are network, fiber distributed data interface metropolitan area network (FDDI MANs), and WANs that use private or public data transmission facilities. DECnet can also use TCP/IP and OSI protocols as well as its own protocols. It is routable protocol.

DECnet has been updated several times; each update is called a phase. V, and the protocols used are both Digital proprietary and a fairly complete implementation of the OSI protocol suite.

APPENDIX B OSI REFERANCE MODEL

In 1978, the International Standards Organization (ISO) released a set of specifications that described a network architecture for connecting dissimilar devices. The original document applied to systems that were open to each other because they could all use the same protocols and standards to exchange information.

In 1984, the OSI released a revision of this model and called at the Open System Interconnection (OSI) reference model. The 1984 revision has become an international standard and servers as a guide for networking.

The OSI model is an architecture that divides network communication into seven layers. Each layer covers different network activities, equipment, or protocols.

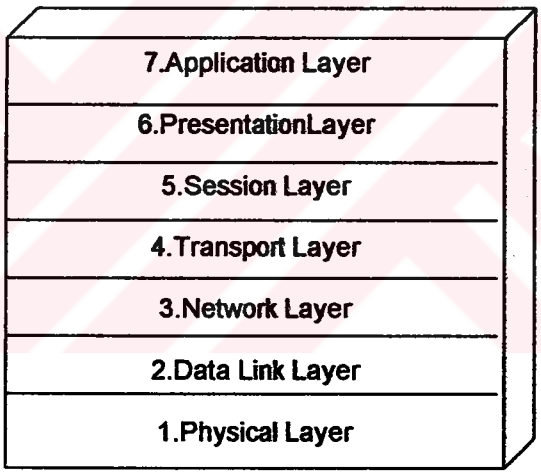

Figure B.1. The seven–layer OSI model

The purpose of each layer is to provide services to the next higher layer and shield the upper layer from the details of how the services are actually implemented. The layers are set up in such ha way that each layer acts as if it is communicating with its associated layer on the other computer. This is a logical or virtual communication between peer layers. In reality, actual communication takes places between adjacent layers on one computer. At each layer there is software that implements certain network functions according to a set protocols.

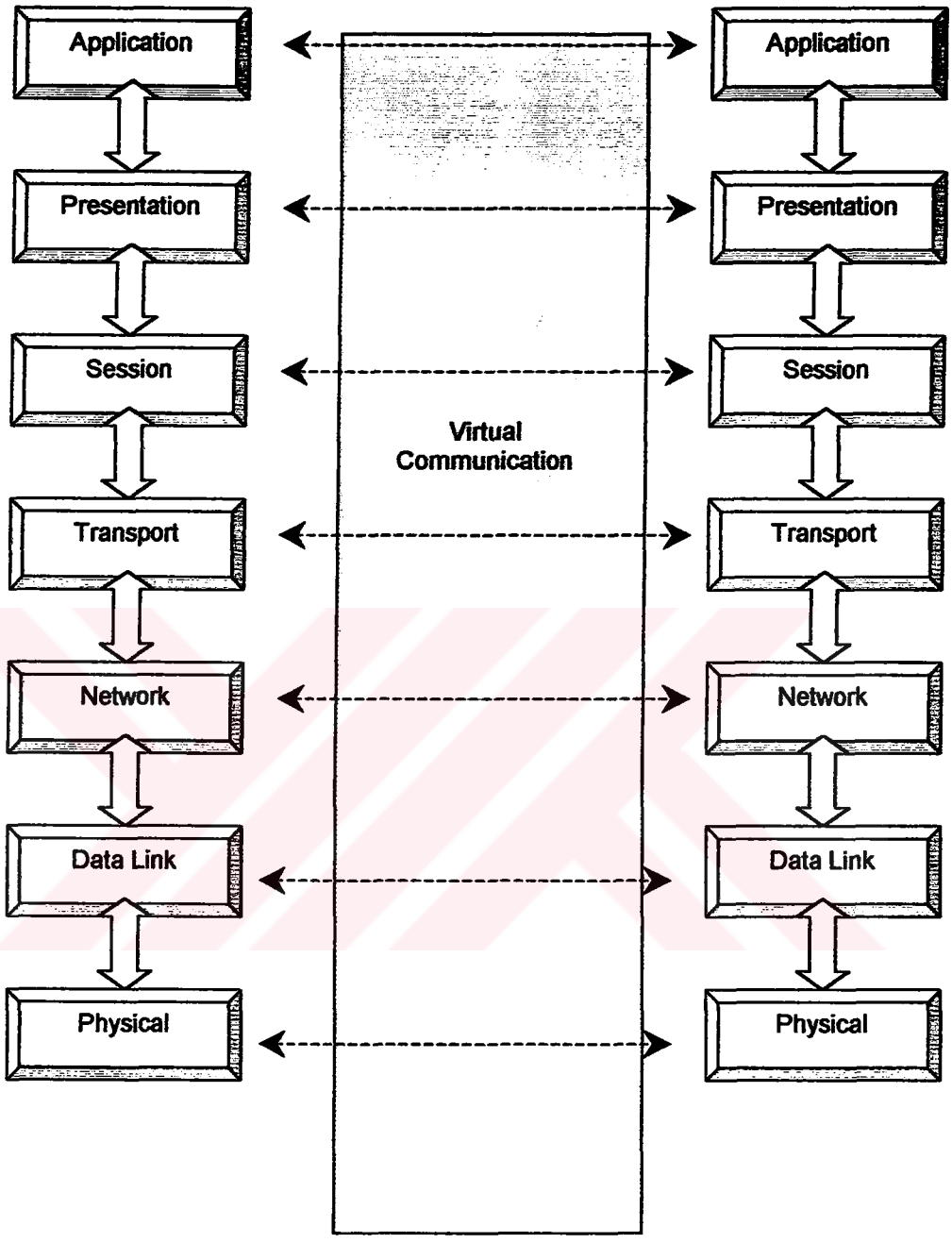

Figure B.2. Relationship among OSI layers

Before data is passed from one layer to another it is broken into packets. A packet is a unit of information transmitted as a whole from one device to another on a network. The network passes a packet from one software layer to another in the order of the layers. At each layer the software adds some additional formatting or

addressing to the packets, which it needs to be successfully transmitted across the network.

At the receiving end, the packet passes through the layers in the reverse order. Software utility at each layer reads the information on the packet, strips it away, and passes the packet up to the next layer. When the packet finally gets passed up to the Application layer, the addressing information has been stripped away and the packet is in its original form, which is readable by the receiver.

Except for the lowest layer in the networking model, no layer can pass information directly to its counterpart on another computer. Information on the sending computer must be passed through all for the lower layers. The information then moves across the networking cable to the receiving computer and up that computer's networking layers until arriving at the same level that sent the information on the computer A, it the cable, and up the Physical and Data Link layer on the receiving side to its destination at the Network layer on computer B.

The following sessions describe the purpose of the each of the seven layers of the OSI model and identify services that they provide to adjacent layers.

1. Application Layer

Layer 7, the topmost layer of the OSI model, is the Application layer. It serves as the window for application processes to access network services. This layer represents the services that directly support user applications, such as software for file transfers, for database access, and for e-mail. The lower level support these task performed at the application level. The Application level handles general network access, flow control, and error recovery.

2. Presentation Layer

Layer 6, the Presentation layer, determines the format used to exchange data among networked computers. It can be called the network's translator. At the sending computer, this layer translates data from a format sent down from the

Application layer into a commonly recognized, intermediary format. At the receiving computer, this layer translates the intermediary format into a format useful to that computer's Application layer. The presentation layer is responsible for protocol conversion, translating the data, encrypting the data, changing or converting the character set, and expanding graphics commands. The presentation layer also manages data compression, to reduce the number of bits that need to be transmitted.

A utility known as the redirector operates at this layer. The purpose of the redirector is to redirect input/output (I/O) operations to resources on a server.

3. Session Layer

Layer 5, the Session layer, allows two applications on different computers to establish, use, and end a connection called a session. This layer performs name recognition and the functions, such as security, needed to allow two applications to communicate over the network.

The Session layer provides synchronization between user tasks by placing checkpoints in the data stream. This way, if the network fails, only the data after the last checkpoint has to be retransmitted. This layer also implements dialog control between communicating processes, regulating which side transmits, when, for how long, and so on.

4. Transport Layer

Layer 4, the Transport layer, provides an additional connection level beneath the Session layer. The Transport layer ensures that packets are delivered error free, in sequence, an with no losses or duplications. This layer repackages messages, dividing long messages into several packets and collecting small packets together in one package. This allows the packets to be transmitted efficiently over the network. At the receiving end, the Transport layer unpacks the messages, reassembles the original messages, and typically sends an acknowledgment of receipt.

The Transport layer provides flow control, error handling, and is involved in solving problems concerned with the transmission and reception of packets.

5. Network Layer

Layer 3, the Network layer, is responsible for addressing messages and translating logical addresses and names into physical addresses. This layer also determines the route from the source to the destination computer. It determines which part the data should take based on network condition, priority of services, and other factors. It also manages traffic problems on the network, such as packet switching, routing, and controlling the congestion of data.

If the network adapter on the router cannot transmit a data chunk as large as the source computer sends, the Network layer on the router compensates by breaking the data into smaller units. On the destination end, the Network layer reassembles the data.

6. Data Link Layer

Layer2, the Data Link layer, send data frames from the Network layer to the Physical layer. On the receiving end, it packets raw bits from the physical layer into data frames. A data frame is an organized, logical structure in which data can be placed.

Figure A.3 shows an example of a simple data frame. In this example, the sender ID represent the address of the computer that is sending the information.; the destination ID represented the address of the computer to which the information is being sent. The control information is used for frame type, routing, and segmentation information. The data is the information itself. The cyclical redundancy check, (CRC) represents error correction and verification information to ensure the data frame is received. The Data Link layer is responsible for providing the error-free transfer of these frames from one computer to another through the Physical layer.

This allows the Network layer to assume virtually error-free transmission over the network connection.

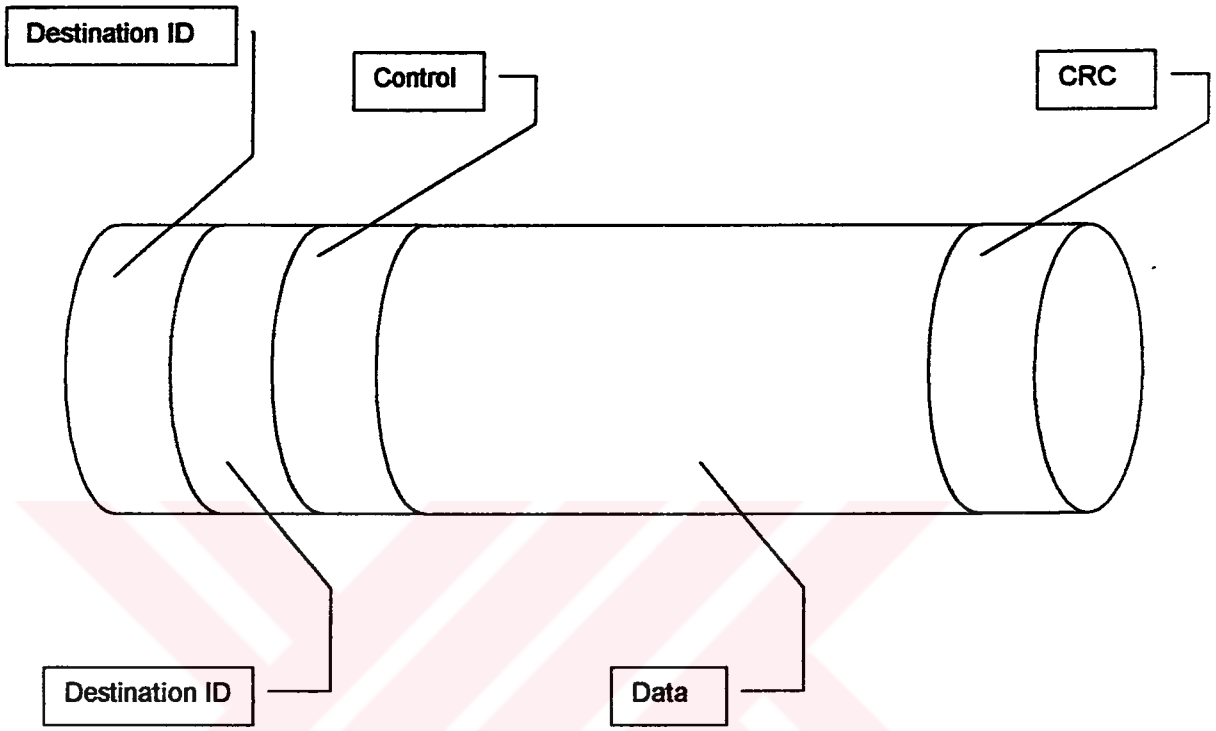

Figure B.3. A simple data frame

Generally, when the Data Link layer sends a frame, its waits for an acknowledgement from the recipient. The recipient Data Link layer detects any problem with the frame that may have occurred during transmission. Frames that were no acknowledged, or frames that were damaged during transmission, is resent.

7. Physical Layer

Layer 1, the bottommost layer of the OSI model, is the Physical layer. This layer transmits the unstructured raw bit stream over a physical medium (such as the network cable). The Physical layer relates the electrical, optical, mechanical, and

functional interfaces to the cable. The Physical layer also carries the signals that transmit data generated by all of the higher layers.

This layer defines how the cable is attached to the network adapter card. For example, it defines how many pins the connector has and each pin's function. It also defines which transmission technique will be used to send data over the network cable.

The Physical layer is responsible for transmitting bits (zeros and ones) from one computer to another. The bits themselves have no defined meaning at this level. This layer defines data encoding and bit synchronization, ensuring that when a transmitting host sends a 1 bit, it is received as a 1 bit, not a bit 0 bit. This layer also defines how long each bit lasts and how each bit is translated into the appropriate electrical or optical impulse for the network cable.

APPENDIX-C OCX's USAGE

An OCX is an Object Linking and Embedding (OLE) custom control, a special-purpose program that can be created for use by applications running on Microsoft's Windows systems. OCXs provide such functions as handling scroll bar movement and window resizing. If you have a Windows system, you'll find a number of files in your Windows directory with the OCX file name suffix.

Object Linking and Embedding was designed to support compound documents (which contain multiple information types, such as text, graphic images, sound, motion video). The Windows desktop is an example of a compound document and Microsoft used OLE to build it. OLE and the Component Object Model (COM), a more general concept that succeeded OLE, support the development of "plug-and-play" programs that can be written in any language and used dynamically by any application in the system. These programs are known as components and the application in which they are run is known as a container. This component-based approach to application development reduces development time and improves the program capability and quality. Windows 95 and NT application development programs such as Powerbuilder and Microsoft Access take advantage of OCXs.

1. Winsock OCX

1.1 Properties:

BytesReceived Property, Index Property, *LocalHostName* Property, *LocalIP* Property, *LocalPort* Property, Name Property, Object Property, Parent Property, Protocol Property, *RemoteHost* Property, *RemoteHostIP* Property, *RemotePort* Property, *SocketHandle* Property, State Property, Tag Property (http://www.socket.com).

1.2. Events:

Close Event, Connect Event, *ConnectionRequest* Event, *DataArrival* Event, Error Event, *SendComplete* Event, *SendProgress* Event,

1.3. Methods:

Accept Method, Bind Method, Close Method (Winsock Control), Listen Method, *PeekData* Method, *SendData* Method, *GetData* Method (Winsock Control).

1.4. Accept Method:

For TCP server application only, this method is used to accept an incoming connection when handling a *ConnectionRequest* event.

The accept method is used in the *ConnectionRequest* event. The *ConnectionRequest* event has a corresponding argument, the *RequestID* parameter that should be passed to the Accept method.

1.5. Bind Method:

Specifies the *LocalPort* and *LocalIP* to be used for TCP connections. Use this method if you have multiple protocol adapters. You must invoke the **Bind** method before invoking **Listen** method.

1.6. BytesReceived Property

Returns the amount of data received (currently in the receive buffer). Use the *GetData* method to retrieve data.

1.7. Close Event

Occurs when the remote computer closes the connection. Application should use the Close method to correctly close a TCP connection.

1.8. Close Method (Winsock Control)

Close a TCO connection or listening socket for both client and server application.

1.9. Connect Method

Request a connection to a remote computer.

1.10. ConnectionRequest Event

Occur when the remote machine requests a connection for TCP server application only. The event is activated when there is an incoming connection requests. *RemoteHostIP* and *RemotePort* properties store the information about the client after the event is activated. The server can decide whether or not to accept the connections. If the incoming connection is not accepted, the peer (client) will get the *Close* event. Use the *Accept* method (on a new control instance) to accept an incoming connection.

1.11. DataArrival Event

Occurs when new data arrives. This event will not occur if you do not retrieve all the data in one *GetData* call. It is activated only when there is new data. Use the *BytesReceived* property to check how much data is available at any time.

1.12. Error Event

Occurs whenever an error occurs in background processing (for example, failed to connect, or failed to send or receive in the background).The settings for number are:

Constant	Value	Description
SckOutOfMemory	7	Out of memory
SckInvalidpropertyValue	380	The property value is invalid
SckGetNoSupported	394	The property can't be read
SckSetNoSupported	383	The property is read-only
SckBadState	40006	Wrong protocol or connection state for the requested transaction or request
SckInvalidArg	40014	The argument passed to a function was not in the correct format or in the specified range
SckSucces	40017	Successful
SckUnsupported	40018	Unsupported variant type.
SckInvalidOp	40020	Invalid operation at current state
SckOutOfRange	40021	Argument is out of range
SckWrongrotocol	40026	Wrong protocol for the requested transaction or request
SckOpCanceled	1004	The operation was canceled
SckInvalidArgument	10014	The requested address is a broadcast address, but flag is not set.
SckWouldBlock	10035	Socket is non-blocking and the specified operation will block.
SckInProgress	10036	A blocking Winsock operation in progress.
SckAlreadyComplete	10037	The operation is completed. No blocking operation in progress
SckNotSocket	10038	The descriptor is not a socket.
SckMsgTooBig	10040a	The datagram is too large to fit into the buffer and is truncated.
SckPortNotSupported	10043	The specified port is not supported.
SckAddressInUse	10048	Address in use.
SckAddressNotAvailable	10049	Address not available from the local machine.

SckNetworkSubsystemFailed	10050	Network subsystem failed
SckNetworkUnreachable	10051	The network cannot be reached from this host at this time.
SckNetReset	10052	Connection has timed out when SO_KEEP ALIVE is set.
SckConnectAborted	11053	Connection is aborted due to timeout or other failure.
SckConnectionReset	10054	The connection is reset by remote side
SckNoBufferSpace	10055	No buffer space is available
SckAlreadyConnected	10056	Socket is already connected
SckNotConnected	10057	Socket is not connected
SckSocketShutdown	10058	Socket has been shut down.
SckTimedout	10060	Socket has been shut down.
SckConnectionRefused	10061	Connection is forcefully rejected.
SckNotInitialized	10093	WinsockInit should be called first.
SckHostNotFound	11001	Authoritative answer: Host not found.
SckHostNotFoundTryAgain	11002	Non-Authoritative answer: Host not found.
SckNonRecoverableError	11003	Non-recoverable errors.
SckNoData	11004	Valid name, no data record of requested type

Table C.1. Error Socket Constant

1.13. GetData Method

Retrieves the current blocks of data and stores it in a variable of type variant. It's common to use the *GetData* method with the *DataArrival* event, which includes the *totalBytes* argument. If you specify a *maxlen* that is less than the *totalBytes* argument, you will get the warning 10040 indicating that the remaining bytes will be lost.

The settings for type are:

Description	Constant	Description	Constant
Byte	vbByte	Currency	vbCurrency
Integer	vbInteger	Date	vbDate
Long	vbLong	Boolean	vbBoolean
Single	vbSingle	SCODE	vbError
Double	vbDouble	String	VbString
Byte Array	vbArray+vbByte		

Table C.2. Data Type

1.14. Listen Method

Creates a socket and sets it in listen mode. This method works only for TCP connections. The *ConnectionRequest* event occurs when there is an incoming connection. When handling *ConnectionRequest*, the application should use the Accept method (on new control instance) to accept the connection.

1.15. LocalHostName Property

Returns tl1e local machine name. Read-only and unavailable at design time.

1.16. LocalIP Property

Returns the IP address of the local machine in the IP address dotted string format (xxx.xxx.xxx.xxx). Read-only and unavailable at design time.

1.17. LocalPort Property

Returns or sets the local port to use. Read/write and available at design time.

For the client, this designates the local port to send data from. Specify port 0 if the application does not need a specific port. In this case, the control will select a random port, after a connection is established, this is the local port used for the TCP connection.

For the server, this is the local port to listen on. If port O is specified, a random port is used. After invoking the Listen method, the property contains the actual port that has been selected.

Port 0 is often used to establish connections between computers dynamically. For example, a client that wishes to be "called back" by a server can use port 0 to procure a new (random) port number, which can then be given to the remote computer for this purpose.

1.18. PeekData Method

Similar to *GetData* except *PeekData* does not remove data from the input queue. This method works only for TCP connections.

The settings for type are:

Description	Constant	Description	Constant
Byte	VbByte	Currency	VbCurrency
Integer	VbInteger	Date	vbDate
Long	VbLong	Boolean	vbBoolean
Single	VbSingle	SCODE	vbError
Double	VbDouble	String	VbString
Byte Array	vbArray+vbByte		

Table C.3. Equivalent to Visual Basic™

1.19. Protocol Property (Winsock Control)

Returns or sets the protocol, either TCP or UDP, used by the Winsock Control.

Constant	Value	Description
SckTCPProtocol	0	Default. TCP protocol.
SckUDPProtocol	1	UDP protocol

Table C.4. Winsock Control Property

1.20. RemoteHost Property (ActiveX Controls)

Returns or sets the remote machine to which a control sends or receives data. You can either provide a host name, for example, "www.faraday.cu.edu.tr" or an IP address string in dotted format, such as "193.255.196.66". When this property is specified, the URL property is updated to show the new value. Also, if the host portion of the URL is updated, this property is also updated to reflect the new value. The *RemoteHost* property can also be changed when invoking the *OpenURL* or Execute methods. At run time, changing this value has no effect until the next connection.

1.21. SendComplate Event

Occurs when a send operation is completed.

1.22. SendData Events

Sends data to a remote computer for binary data, byte array should be used.

1.23. SendProgress Event

Occurs while data is being sent. The SendProgress syntax is

Winsock_SendProgress(bytesSend as Long, bytesRemaining as Long)

Part	Description
bytesSend	The number of bytes that have been sent since the last time this event was activated.
bytesRemaining	The number of bytes in the send buffer waiting to be sent.

1.24. SocketHandle Property

Returns a value that corresponds to the socket handle to the control uses to communicate with the **Winsock** layer. Read-only and unavailable at design time.

1.25. State property

Returns the state of the control, expressed as an enumerated type. Read-only and unavailable at design time. The settings for the State property are:

Constant	Value	Description
sckClosed	0	Default Closed
sckOpen	1	Open
scklistening	2	Listening
sckConnectionpending	3	Connection pending
sckResolvingHost	4	Resolving host
sckHostResolved	5	Host resolved
sckConnecting	6	Connecting
sckConnected	7	Connected
sckClosing	8	Peer is closing the connection
sckError	9	Error

APPENDIX-D WINDOWS DLLs

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function SystemParametersInfo Lib "user32" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uParam As Long, ByRef lpvParam As Any, ByVal fuWinIni As Long) As Long

Public Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function FindWindowEx Lib "user32" Alias "FindWindowExA" (ByVal hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long

Public Declare Function GetWindowTextLength Lib "user32" Alias "GetWindowTextLengthA" (ByVal hwnd As Long) As Long

Public Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long

Public Declare Function GetWindowText Lib "user32"Alias "GetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long

Public Declare Function ShowWindow Lib "user32"(ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Public Declare Function IsIconic Lib "user32" (ByVal hwnd As Long) As Long

Public Declare Function IsZoomed Lib "user32" (ByVal hwnd As Long) As Long

Public Declare Function IsWindowVisible Lib "user32" (ByVal hwnd As Long) As Long

Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long)

Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long)

Public Declare Function ExitWindowsEx& Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long)

Private Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal dwError As Long, ByVal lpstrBuffer As String, ByVal uLength As Long) As Long

Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength As Long, ByVal hwndCallback As Long) As Long

Private Declare Function GetVolumeInformation Lib "kernel32.dll" Alias "GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Integer, lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long

Private Declare Function GetComputerName Lib "kernel32" Alias "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long

Private Declare Function SendMessage Lib "user32" Alias SendMessageA"_ (ByVal hwnd As Long, ByVal wMsg As Long,ByVal wParam As Long, lParam As Any) As Long

Private Declare Function GetKeyState Lib "user32"(ByVal nVirtKey As Long) As Long

Private Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long

Private Declare Function SetCursorPos Lib "user32" (ByVal x As Long, ByVal y As Long) As Long

Private Declare Sub mouse_event Lib "user32" (ByVal dwFlags As Long, ByVal dx As Long, ByVal dy As Long, ByVal cButtons As Long, ByVal dwExtraInfo As Long)