EXTENDED SERVICES FOR WINDOWS NETWORK OPERATING SYSTEM

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING
OF THE BHARATHIAR UNIVERSITY

P- 499

By

A.PRAVEEN

D.GEETHA

M.SHEEBA RANI

M.K.INDUMATHY

Under the Guidance of

Ms. S. RAJINI. B.E.,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006

2000-2001

Department of Computer Science and Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641006

CERTIFICATE

This is to certify that the report entitled

EXTENDED SERVICES FOR WINDOWS NOS

has been submitted by

Mr. A. Praveen
Ms. D. Geetha
Ms. M. Sheeba Rani
Ms. M. K.Indumathy

in partial fulfillment for the award of the Bachelor of Engineering degree in the Computer Science and Engineering branch of Bharathiar University,Coimbatore-641006 during the academic year 2000-2001.

[illegible] 9.03.2001
Guide

[illegible] 9/3
Head of the Department

Certified that the Candidate was examined by us in the Project work

Viva Voce Examination held on ...12.3.2001.... and

the university register number was.9727K0160.

9727K0137, 9727K0141, 9727177........

Internal Examiner

External Examiner

DECLARATION

We declare that the project work entitled **"Extended services for Windows NOS"** submitted to Bharathiar University, Coimbatore, in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Science, is a record of original work done by us under the guidance of **Ms. S. Rajini, B.E.,** Lecturer, Kumaraguru College of Technology, Coimbatore. The report has not been submitted to any other institute/University for any award of any degree.

Praveen. A.
Geetha. D.
Sheeba Rani. M.
Indumathy. M. K.

CONTENTS

ACKNOWLEDGEMENT

An endeavor over a long period can be successful only with the advice and support of many well wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We express our gratitude to **Dr. K. K. PADMANABHAN, B.Sc. (Engg), M.Tech., Ph.D.,** *Principal, Kumaraguru College of Technology, Coimbatore* for his constant encouragement throughout our course, despite his busy schedule.

We wish to thank **Dr. S. THANGASAMY, Ph.D.,** *Professor, Head of the Department of Computer Science and Engineering, Kumaraguru College Of Technology, Coimbatore* for his extensive support and encouragement.

Our heartfelt thanks to our project guide **Miss. S. RAJINI, B.E.,** *Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore* for her constant guidance and unfailing interest in aiding us to consummate this project successfully.

Our special and sincere thanks to **Mr. M. N. GUPTHA, B.E.,** *Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore* who have been a superb source of inspiration, guidance and encouragement. We thank him for all the help he rendered despite his busy schedule.

We are deeply indebted to **Miss. A. LAVANYA B.E.,** *Lecturer, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore* for inspiring us and enable us to complete the project.

We extend our thanks and gratitude to all the staff members of Computer Science and Engineering Department, Kumaraguru College of Technology for their best support and guidance provided to us.

We also would like to express our sincere thanks to all our classmates and friends who were always there when the need arose.

Last but not the least we would like to thank our family for their support and constant encouragement, which was instrumental in our successful completion of this project.

SYNOPSIS

The project entitled as **"EXTENDED SERVICES FOR WINDOWS NETWORKING OPERATING SYSTEM"** is the application created using Windows programming with the aid of Visual C++. It provides the enchanced services for Windows NOS.

Windows is not suitable for computing intensive applications which can be hosted on the Microsoft Windows NT Operating System.Windows NT allows the networked users to access file servers to store information and print servers to print documents. By understanding the basic architecture of NT environment, we should be able to design an important and efficient Windows and Windows NT application that can honour the power of network computing.

We develop some of the applications, which enhance the standard features of Windows application. They are explained as follows.

A Standard Application does not provide any privilege to access files from a remote computer. It does not allow any access to remote hard disk, printer etc. But by including this application, it is possible to access other system resources.

One of the fascinating features is Resource sharing, done through the mechanism of mapping. It is also possible to control the systems connected to the network.

Editor is one of the important applications, created for doing various modification operations such as changing fonts, foreground, background etc., and also possible to open, close and save files. Using the editor one can also print a document by making use of print option.

Generally it is not possible to have the pictures created by us as the screen savers. But using this application, it is possible to have personalized screen savers. This can be easily implemented with the help of Visual C++.

1. INTRODUCTION

1.1 Project overview

The project work entitled "**EXTENDED SERVICES FOR WINDOWS NETWORK OPERATING SYSTEM**" facilitates the access of resources which can be shared from the systems connected to the network. This service is implemented by means of windows programming.

1.1.1 Objectives

The main objectives of our project are to create the extended services, which are listed below.

1.Device operations.
2.Editor.
3.Screen saver.
4.File Management.

1.Device operations

Any device connected to the system over the network can be activated and controlled by a single click event from the server. It is possible to print any remote file using a printer connected to the remote node.

2.Editor

A new document can be created and saved. It can be altered to the required format. Any exiting document can be viewed which is present in the remote node and it can also be modified.

3.Screen saver

Normally, any bitmap file cannot be converted into screensaver file. By using our application, bitmap files are converted into screensaver files. These files can also be customized.

4.File Management

It is possible to access a file from a remote node. Some more operations like creating directory, deleting it, modifying any file etc. can also be done. The file is treated, as it is present in the local node.

2. SYSTEM ANALYSIS

2.1 Features of VC++

The Visual C++ Environment

The central part of the Visual C++ packages is developer studio, the integrated development environment (IDE). Developer Studio is used to integrate the development tools and the Visual C++ Compiler. A Windows program can be created scan through an impressive amount of online help and debug a program without leaving Developer Studio.

Developer Studio Tools

Developer Studio includes a number of tools such as

- An integrated editor offers drag and drop and syntax highlighting as two of its major features.
- A resource editor is used to create Windows resources, such as bitmaps, icons, dialog boxes and menus.
- An integrated debugger enables to run programs and check for errors. Because the debugger is part of Developer Studio, it is easy to find and correct bugs.

Developer Studio also features an online help system, which can be used to get context-sensitive help for all of the tools included in Developer Studio, as well as detailed help on the C++ language, the Windows programming interface, and the MFC class library.

Developer Studio Wizards

In addition to tools that are used for debugging, editing and creating resources, Developer Studio includes three wizards that are used to simplify developing your Windows programs:

- *AppWizard* is used to create the basic outline of a Windows program. The AppWizard has a central role in the Visual C++ development system for creating skeleton applications. The AppWizard can be used to create project workspaces for MFC-based Windows applications and Windows DLLs.

AppWizard supports three types of programs: single document and multiple document applications based on the Document/View architecture and dialog box-based programs, in which a dialog box serves as the application's main window.

Multiple Document Interface:

A MDI application can open many documents at once. There is a window menu and a Close item on the File menu.

Single Document Interface:

A SDI application has only one document open at a time.

Dialog-based application:

It does not have a document at all. There are no menus.

- *ClassWizard* is used to define the classes in a program created with AppWizard. Using Class Wizard, you can add classes to your project. You can also add functions that control how messages received by each class are handled. ClassWizard also helps manage controls that are contained in dialog boxes by enabling you to associate an MFC object or class member variable with each control.

- OLE ControlWizard is used to create the basic framework of an OLE control. An OLE control is a customized that supports a defined set of interfaces and is used as a reusable component. OLE controls replace Visual Basic controls, or VBXs, that were used in 16-bit versions of Windows.

Win32

Win32 is a family of Windows programming Interfaces. It is the standard programming interface for Windows 95 and Windows NT. The Win32 Application Programming Interface (API) is specified in terms of 32-bit values used to program Windows. The use of 32-bit address greatly simplifies the task of programming. Programmers work in a "flat" 32-bit address space, instead of the 16-bit "segmented" addresses of the older Windows versions. The Windows NT API supports everything of the Win95 API as well as more powerful graphics capabilities and sophisticated file protection and sharing. Windows NT also supports multiprocessor systems, that is, systems in which there are two or more processors (CPUs) sharing the same physical memory.

Windows NT APIs support powerful controls such as the Rich Text Control that allow us to construct much nicer user interfaces quickly. Like the "common dialogs" introduced so successfully into Windows 3.1,the Win32 "common controls" make it easy to standardized the "look and feel" of your applications.

Win32 also supports a style of control called the " OCX" control. OCX controls are implemented using OLE Automation. Unlike VBX controls, which could only operate under Win16, we can built OCX controls in both 16- and 32-bit versions. Using OCX controls allows us to readily integrate new kinds of control into our application.

MFC Libraries

MFC is a class library that makes programming for Windows much easier. Most of the MFC classes fall into the following major categories.

- Application Architecture includes classes that help provide the basic plumbing for applications written using MFC.
- Dialog Boxes category includes classes that handle the common dialog boxes that are included in Windows 95.
- Views are used in the Document/View architecture to represent the program's output. Classes that support

scrolling, editing and windows based on dialog boxes are included in this category.

- Controls are used to provide easy access to all of the controls offered by Windows 95 and Windows NT. Included in this category are classes that manage tree controls, list views and combo boxes.
- Graphical Objects are used for creating the output in a Windows program. This category includes classes for pens, brushes, icons and bitmaps.
- Exceptions are used to indicate that an unexpected event occurred during a program's execution.
- Collections are a special type of class used to contain objects from another class. Some of the collection classes offered in the MFC class library are template-based, which enables you to use them with almost any object.
- OLE classes are used to provide support for creating OLE-aware applications.
- Miscellaneous classes include classes for strings, rectangles and WOSA (Windows Open System Architecture) services such as socket-based communication and MAPI (Messaging Application Programming Interface).

By using the MFC classes for writing programs for Windows, the user can take the advantage of a large amount of source code that has been written for him. MFC is portable.

Object Window Library (OWL)

The most popular Windows Programming class library is MFC (Microsoft Foundation Classes). Microsoft Corporation develops it. In the case of Borland VC++, OWL is used instead of MFC. OWL contains all classes of library, which are powerful development tools that offer significant advantage in some situations.

Dynamic Link Libraries (DLL)

DLLs are windows-based program modules that can be loaded and linked at runtime. Many applications can benefit by being split into a series of main programs and DLLs. This feature is very helpful for a developing a large application. If all modules share the same list management and database access classes; the shared code can be put in one or more DLLs, the individual modules will be smaller and on disk and therefore quicker to load.

Another use for DLLs is national language support. If language-dependent functions and resources are isolated into DLLs, the user can choose proper DLL during the installation process or at run time.

There are several DLL linkage options. An MFC library DLL can accommodate entire C++ classes. These DLL-resident classes can be used the same way that statically linked classes are used. Objects can be constructed of DLL-resident classes and can be used as base classes.

A DLL is a collection of functions used by many different applications, which makes programs smaller. A regular DLL linked with the MFC DLL can, in turn, be called from any applications.

Dynamic Data Exchange [DDE]

In windows, dynamic data exchange (DDE) is a form of interprocess communications to exchange data between applications. applications can use DDE for one-time data transfers and for ongoing exchanges and updating of data.

Database

MFC supports two different kinds of database access:

- Access via Data Access Objects (DAO) and the Microsoft Jet database engine
- Access via Open Database Connectivity (ODBC) and an ODBC driver.

DAO provides a set of data access objects: database objects, tabledef and querydef objects, recordset objects, and others. DAO is optimally useful for working with .MDB files like those created by Microsoft Access, but you can also access ODBC data sources through DAO and the Microsoft Jet database engine.

ODBC provides an application-programming interface (API) which different database vendors implement via ODBC drivers specific to a particular database management system (DBMS). Your program uses this API to call the ODBC Driver Manager, which passes the calls to the appropriate driver. The driver, in turn, interacts with the DBMS using Structured Query Language (SQL).

Both DAO and ODBC give the ability to write applications that are independent of any particular DBMS.

Wizard

One of the most useful GUI concepts is the wizard. Allowing the application to guide a user through the process of completing a task makes even most complicated actions a breeze. In manyways wizards have revolutionized the modern windows environment.

The purpose of a wizard is to collect and organize the user interface needed to complete the specific task. This is accomplished by creating multiple "PAGE" dialogues using a wizard than it is to move through a collection of individual modal dialogues. Also by breaking the user interface up over multiple pages, functionally related items can be more easily grouped, and the user is never overwhelmed by a large number of settings.

2.2 The Windows Programming Model

Windows programming has the reputation of being difficult to learn. This reputation is partially due to the richness of the programming environments. The sheer number of Windows API functions can easily overwhelm us. There are two significant impediments to learning Windows programming:

1.Windows programs require a different conceptual model of the problem than is used by traditional DOS or Unix shell programming.

2.The tools used to write a program based on this conceptual model do not directly support the concepts used.

The Conceptual Model

Windows applications are programs that react to different forms of user input and provide sophisticated graphical output for the user. Any Window displayed by an application must react to user actions. Menus must pop down and enable selections. Objects in general in Windows must react when manipulated. This encourages a conceptual view of Windows application as a collection of objects. This is exactly the view proposed by object–oriented programming.

In object-oriented programming programmer creates abstract data types. These abstract data types are commonly called *objects* and consist of a data structure and associated functions, commonly called *methods*, that manipulate the data structure. Typically, the data structure of an object is completely unknown outside of the methods it provides to the outside world. This approach, called *data encapsulation*, enables the internal structure of the object to change at will. As long as the external interface provided by its methods remains unchanged, the rest of the program does not need to know what the objects looks like internally or how it implements its functionality.

These concepts apply quite naturally to Windows applications. There are many objects in Windows, including these:

- Pens-objects that have a width, a color, and are used to draw line
- Brushes –objects that have a color and leave a certain pattern when used to paint areas
- Menus and dialog boxes

But the first and most fundamental object is called a "Window".

Windows and Their Associated Window Functions

Window has certain characteristics-its location on the screen, a title, and a size, a menu and so on. Think of these as its physical characteristics. But it also has other characteristics: how it reacts to the notifications of various events. These are its behavioral characteristics. All windows have an associated function, called the *window function* that determines how the window reacts to the notification of an event. The notification itself is called a message.

Windows notifies a window when an event occurs that might affect the window. This is usually described differently. That is, a message is sent to the Window or, depending on your point of view, the window receives a message notifying it of the event. In actuality, Windows calls the function associated with the window, passing information in the arguments of the call that describe the event that has occurred. Windows, their associated window functions, and the messages they receive are, therefore, interrelated.

Windows Queues and the Message Loop

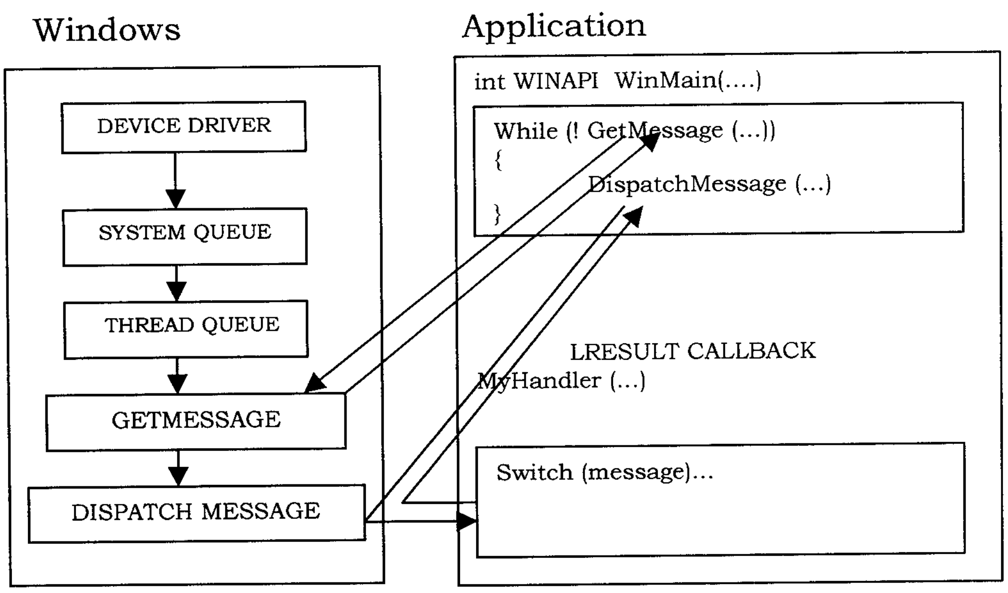
Many messages in Windows originate from devices. Moving the mouse and clicking the mouse and clicking the mouse buttons generate interrupts that are handled by the mouse device driver. These device drivers call Windows to translate the hardware event into a message. The resulting message is then placed into the Windows system queue.

There are two types of queues in Windows: system and thread. There is only one system queue. Hardware events that are converted into messages are placed into the system queue. Messages reside in the system queue only briefly. Each running Windows application has its own unique thread queue.

Windows implements the sharing of the shared resources by using the system queue. When an event occurs, a message is placed into the system queue. Windows then must, in effect, decide which thread queue should receive the message.

The message loop retrieves input messages from the application queue and dispatches them to the appropriate window functions. The message loop continually retrieves and dispatches messages until it retrieves a special message that signals that the loop should terminate. One message loop is the main body of a Windows application. A Windows application initializes, repeatedly executes the message loop logic until instructed to stop, and then terminates.

A program calls the Windows **GetMessage** function to retrieve a message from its thread queue. The program calls the Windows **DispatchMessage** function, inorder to send message to the proper window function.


Windows
Application
DEVICE DRIVER
SYSTEM QUEUE
THREAD QUEUE
GETMESSAGE
DISPATCH MESSAGE
int WINAPI WinMain(....)
While (! GetMessage (...))
{
DispatchMessage (...)
}
LRESULT CALLBACK MyHandler (...)
Switch (message)...

2.3 Features of Windows NT

- Windows NT is a portable operating system able to span several diverse hardware platforms.
- It can be easily extended and enhanced as hardware evolves.
- It is a full-fledged 32-bit operating system.
- It can run on computer with multiple CPUs.
- It provides a secure environment that meets the DOD C2 security classification.

Windows NT contains emulators that allow it to automatically execute programs written for the following operating systems:

- Windows 3.1
- DOS
- OS/2
- POSIX(Portable Operating System Interface based on UNIX)

Windows NT-Client Server Model

It has two modes of execution.

1. User mode.
2. Kernel mode.

1. *User mode*

Application programs run in this mode. It interacts with the operating system interface.

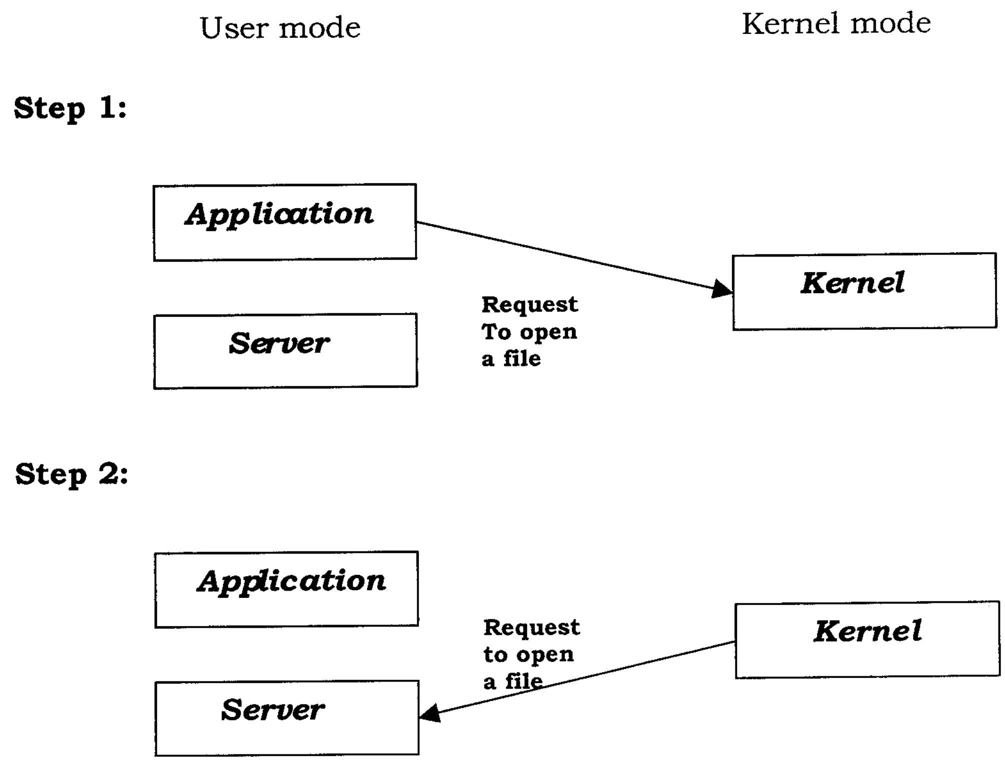
2. *Kernel mode*

In a layered operating system, the entire operating system runs in Kernel mode. This means that all system services run in kernel mode. User mode has access to the hardware and the low-level system services only through the kernel interface.

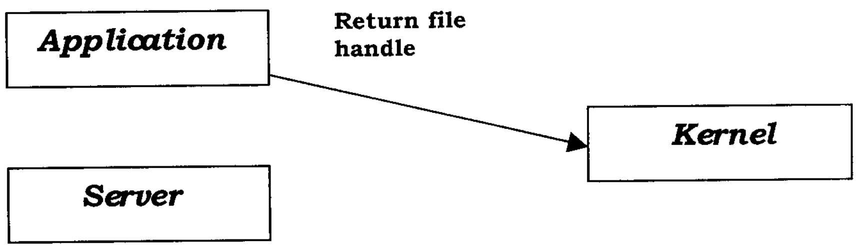
Windows NT breaks from the layered model in the following way:
It moves many of the operating system services out of the kernel. So, the kernel for Windows NT is small.

The Client/Server Model

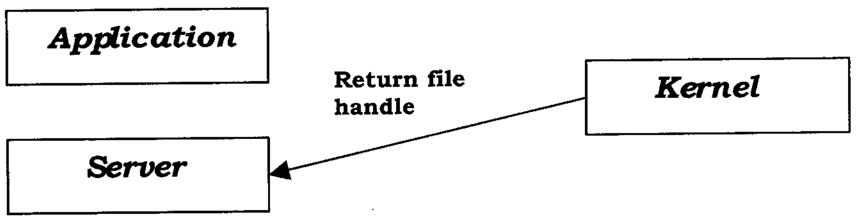
Opening a file



Step 3:



Step 4:



The client/server approach is based on the passing of messages between the application program and the servers it uses. The only way that an application program (i.e., the client) can access a system service (i.e., the server) is to pass a message to the kernel. The kernel then passes this message to the appropriate server, which processes the message and sends a response back to the kernel. The kernel then returns the information to the application program.

The Windows NT Architecture

Applications supported by Windows NT and the protected sub systems run in user mode. The sub systems labeled CSR is the

Client/ Server runtime systems. It handles 32 bit Windows programs.

Architecture of Windows NT 4 is shown in the figure 2.3.1.

Windows NT 4 Architecture

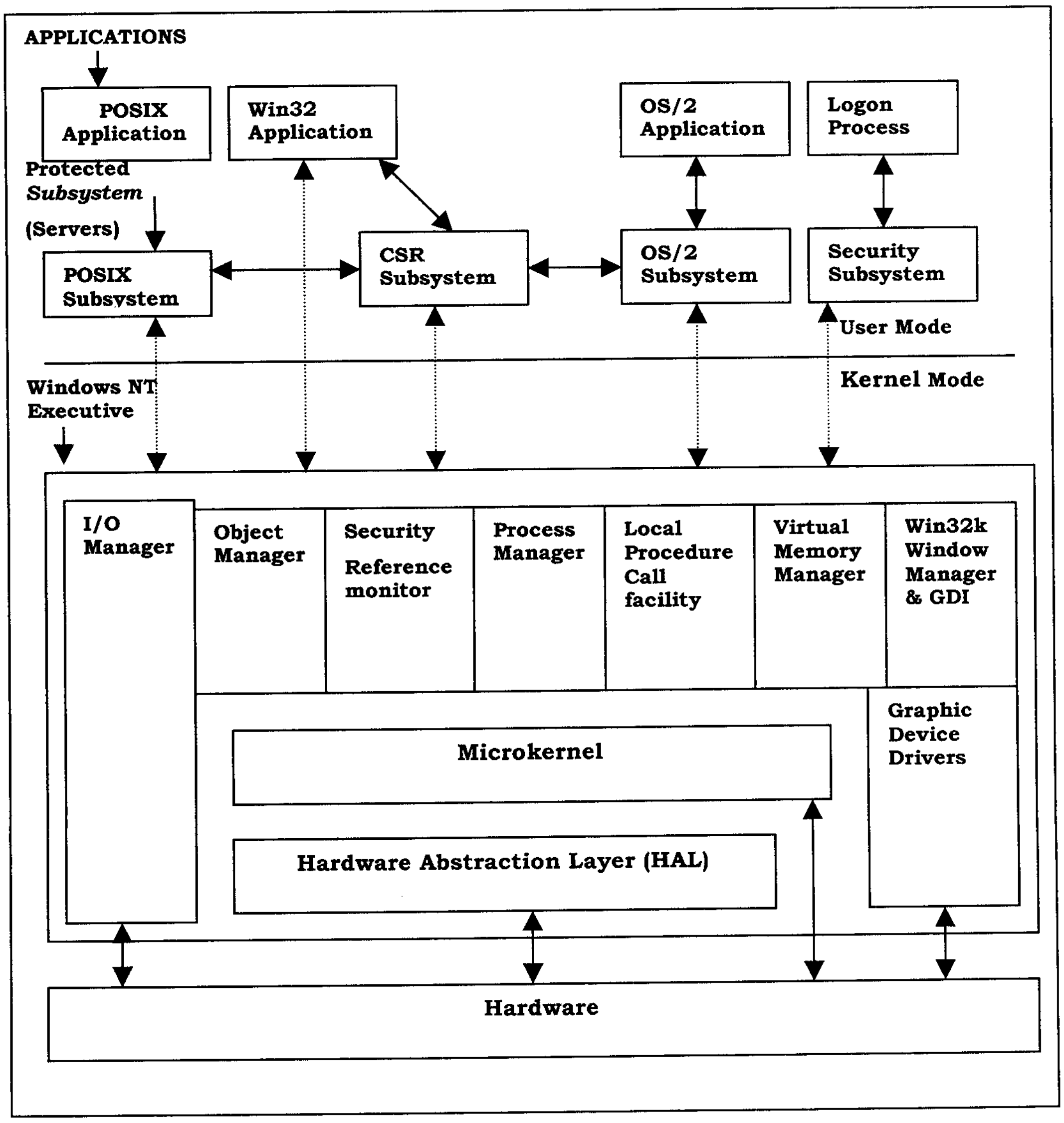


Figure 2.3.1

The Windows NT Executive

The Part of Windows NT that runs in kernel mode is called Windows NT Executive, or NT Executive, for short. It contains the following items:

- The Process Manager
- The Object Manager
- The Hardware Abstraction Layer
- The I/O Manager
- The Security Subsystem
- The Local Procedure Call Subsystem
- The Win32k Executive
- The Kernel itself (also called micro kernel)

The process manager manages the execution tasks and threads in the system. The object manager handles various system resources. The Hardware Abstraction Layer (HAL) contains code that interfaces with the hardware itself. The Local Procedure Call (LPC) subsystem manages the passing of messages between applications and servers as just described. I/O, Security and Virtual Memory are handled by their respective components. The Win32k executive is a new addition to the NT Executive that has been added by Windows NT 4.Within the NT Executive, the Hardware Abstraction Layer and the Kernel are the foundation upon which the other subsystems are built.

Windows NT Programming

There are two ways in programming for Windows NT.

1.API functions defined by win32.

2.MFC Programming.

Advantages of using API functions over MFC

1.Debugging is easy.

2.It provides complete control over how the program executes.

3.All Windows NT programming environment suspends it.

Differences between Normal Programming and Windows Programming

1.There is standard output device in normal programming. But this concept is avoided in Windows Programming. Here screen becomes a shared resource. Each program gives its output to its own window only.

2.There are many built-in functions called as API functions. The codes for their functions are located in **DDL (Dynamic Link Library).** Whenever any one of the API functions is called windows loads and executes the corresponding code through a mechanism called **"Dynamic Library"**.

3.Windows actually send messages to an executing Windows program, which informs it about various activities taking place within the system. Any event, which affects the application, is automatically notified through these messages.

4.All Windows NT programs must contain a special function called window function that is not called by our program but is called by Windows NT. It is through this function that Windows NT communicates with our program. This function is called **"CALLBACK"** function.

2.4 PROPOSED MODEL

2.4.1 Printing

Programming for printing is one of the more complex tasks programmers face when writing Windows applications. The overview provides you with a high-level conceptual view of the Windows printing process. We also created a DLL that we can incorporate into our own applications. It allows us to specify call-backs for formatting headings, footings, and each 'line' of output.

The DLL provides for all the necessary setup of the printing task after we have used the common setup dialogs to choose a printer and a page setup.

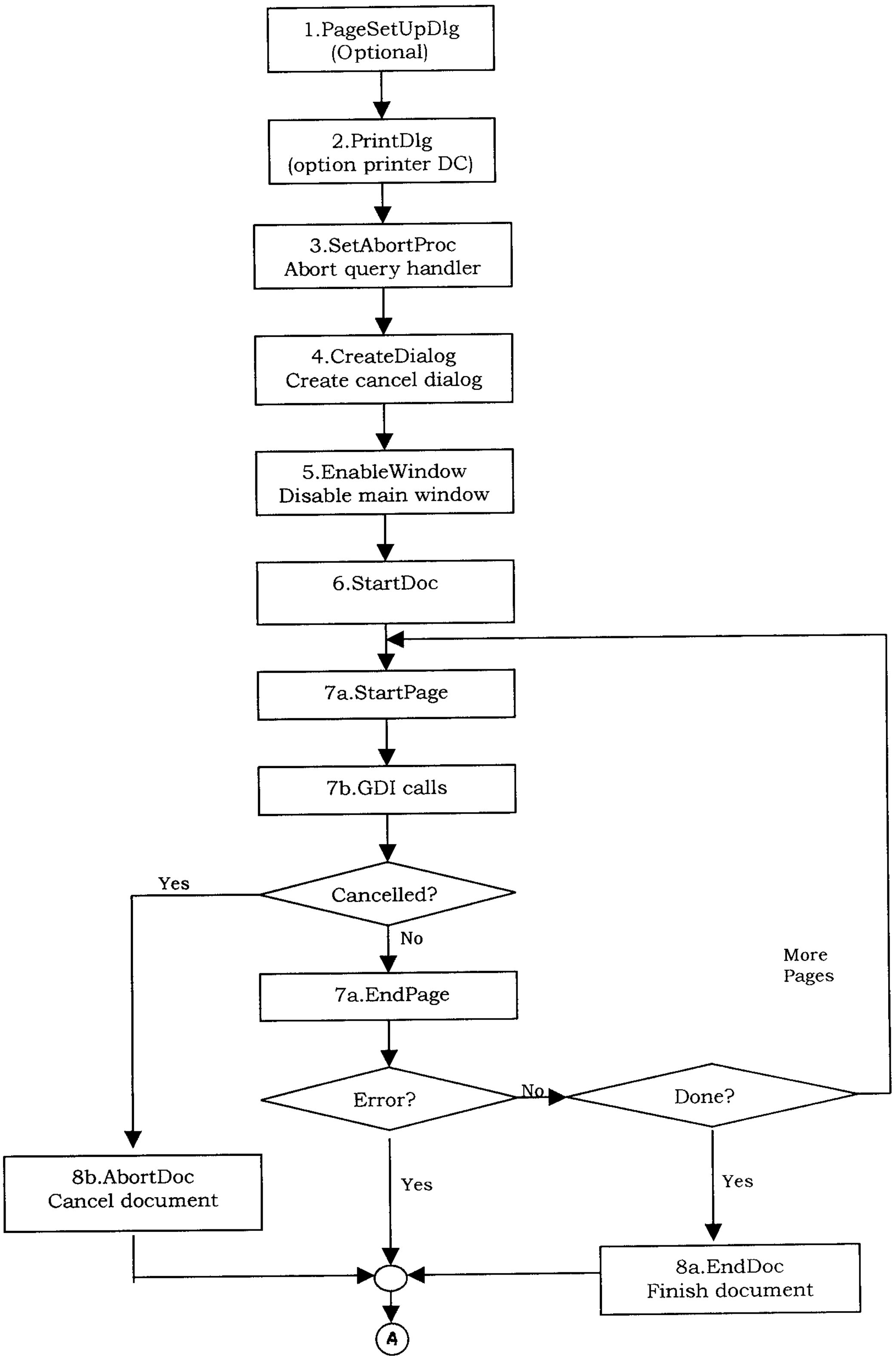
An application must complete a number of steps to print in the Windows environment. We must select a Printer. We must obtain a DC for the printer. We must open and close the *document* and for each page we must open and close each *page*. There are some "housekeeping" functions that allow us to terminate the document and that allow us to tell Windows how to query the user for cancellation. All of these functions are summarized in Table-A.

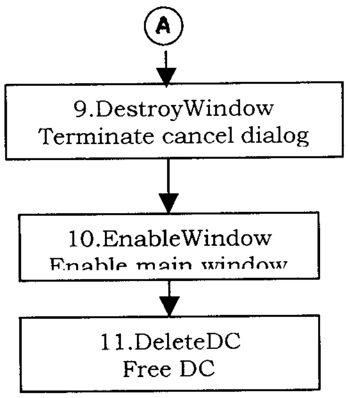
The flowchart of the printing logic gives us the detailed description how each component relates to the printing process. When appropriate, a menu item for Page Setup Dialog may be provided that allows the user to choose page margins, paper size, and other such features for the printer. It is to be decided whether the application should print using the current printer or give the user the ability to select one of the remote printers. The PrintDlg function is used to obtain a DC for the printer.

TABLE-A

FUNCTIONS	**DESCRIPTION**
AbortDoc	Terminates a print job.
EndDoc	Ends a print job.
EndPage	Ends a page.
PageSetupDlg	Allows the user to configure page setup, such as choosing the margin sizes, paper orientation, paper size, and the like. Optional.
PrintDlg	Allows the user to set up and configure printing options. It's most important usage in to obtain a DC for printing.
SetAbortProc	Establishes a procedure that is called to check for user termination of the printing process.
StartDoc	Starts a print job.
StartPage	Notifies the printer driver to prepare to receive output for a new page.
ResetDC	Updates a DC, thereby allowing new functionality that was not supported by any escape function under previous versions of Windows. For example, we can use this function to change orientation or paper bins in the middle of a print job.

The below diagram gives the Printing logic





2.4.2 Screen Saver

Screen Savers are one of the few smaller applications that make use of the registry. Although screen savers are some of the simpler Windows NT applications, they are also some of the most interesting from the programmers point of view.

While screen savers were initially invented to prevent phosphor burn on idle screens, they have taken on a life of their own. Today most screen savers provide either an entertaining message, an interesting graphics display, a company logo, or a humorous animated sequence. Creating our own screen saver is one of the easier Windows NT programming task.

The screen saver designed here is one which does not use registry. Its purpose is to introduce the basic elements common to all screen savers. The program simply illustrates the mechanics involved in creating a screen saver.

Screen Saver Fundamentals

A screen saver is actually one of the easiest Windows NT applications to develop. One of the reasons for this is that it does not create a main window. Instead, it uses the desktop as its window. It also does not contain a WinMain () function or need to create a message loop. In fact, a screen saver requires only three functions, two of which may be empty placeholders.

When we create a screen saver, our program must include the header file SCRNSAVE.H and we must include SCRNSAVE.LIB when linking. The screen saver library provides the necessary support for screen savers.

The Screen Saver Functions

The three functions that every screen saver must provide are shown below:

Function	***Purpose***
ScreenSaverProc()	This is the screen saver's window procedure. It is passed messages and must respond appropriately.
ScreenSaverConfigureDialog()	This is the dialog function for the screen saver's configuration dialog box. It can be empty if no configuration is supported.
RegisterDialogClasses()	This function is used to register custom class types. It will be empty if no custom classes are used.

Two Screen Saver Resources

All screen savers must define two special resources: an icon and a string. The icon, whose ID must be ID_APP, is used to identify the screen saver. The string resource, whose identifier must be IDS_DESCRIPTION, contains a description of the screen saver.

2.4.3 Editor

The Text Editor implemented has the following options:

This editor can be activated in two modes.

1. Using Keyboard - shortcut keys
2. Using Mouse – clicking on the command

The editor is an important tool that is used to perform many useful operations, which can be explained as follows.

We can create a new text document by clicking the 'New' option in the menu list. After typing the needed text, the editing operations such as cut, copy, paste, select all, font, color can be done. It is easy to have the text in the required format.

The formatted text can be easily saved using the 'Save' option listed in the menu. Clicking the 'Open' option can open an existing file even if the file resides in any remote computer. An interesting feature of this is that printing option is also included in editor so that a text document can be printed.

3. PROGRAMMING ENVIRONMENT

3.1 Hardware Configuration

To design and test the application the following hardware is required.

- Pentium II running Windows NT
- 64 MB RAM
- SVGA Color Monitor
- Printer

3.2 Software Configuration

Software Used

To design the application the following software are used.

Operating System

- Windows NT 4.0

Development Tool

- Visual C++

4. SYSTEM DESIGN

4.1 Source Code

```
/*   Editor Program   */


#include<windows.h>
#include<string.h>
#include<commdlg.h>
#define NUMLINES 1

HINSTANCE hc;
DOCINFO docinfo;
PRINTDLG printdlg;
char szWinName[] = "My Win";
int x=0,y=0;
int maxx,maxy;
HDC memDC;
HBITMAP hBit;
HBRUSH hBrush;

void PrintInit(PRINTDLG *printdlg,HWND hwnd);

LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);
BOOL CALLBACK DlgProc(HWND,UINT,WPARAM,LPARAM);
char cm[30];
int PASCAL WinMain(HINSTANCE hThisInst,HINSTANCE
hPrevInst,LPSTR                    lpszArgs,int nWinMode)
{
        hc=hThisInst;
        strcpy(cm,lpszArgs);
        HWND hwnd;
        MSG msg;
        WNDCLASSEX wcl;
if(!hPrevInst)
{
        wcl.cbSize = sizeof(WNDCLASSEX);
        wcl.cbClsExtra=0;
        wcl.cbWndExtra=0;
        wcl.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
        wcl.hCursor=LoadCursor(NULL,IDC_ARROW);
        wcl.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        wcl.hIconSm=LoadIcon(NULL,IDI_APPLICATION);
        wcl.hInstance=hThisInst;
        wcl.lpfnWndProc=WindowProc;
        wcl.lpszClassName=szWinName;
        wcl.lpszMenuName="mymenu";
        wcl.style=0;

 RegisterClassEx(&wcl);
}

        hwnd=CreateWindow(szWinName,"window",
                WS_OVERLAPPEDWINDOW|WS_VISIBLE,
                10,10,500,500,0,0,hThisInst,0);

        while(GetMessage(&msg,0,0,0))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
        return msg.wParam ;
}
LRESULT CALLBACK WindowProc(HWND hwnd,UINT message,WPARAM
wParam,LPARAM lParam)
{

HDC hdc;
TEXTMETRIC tm;
char str[80];
int response;

int copies;

static OPENFILENAME ofn;static char file[60]="";static char
tit[20]="";static char buf[32768]="";
        static HFILE hfile;static long len;static int mes;static FARPROC fp;
        static HWND he;static BOOL flag=TRUE;int i;static int
        s=strlen(cm)-1;

        switch(message)
        {
        case WM_CREATE:

        he=CreateWindow("edit","",WS_VSCROLL|ES_MULTILINE|
        WS_HSCROLL|WS_CHILD|WS_VISIBLE|WS_BORDER,0,0,0,0,
        hwnd,(HMENU)10,hc,0);
```

```
SetFocus(he);
        if(cm[0]=='"')
        {
        for(i=0;i<s;i++)
                cm[i]=cm[i+1];
        cm[s-1]='\0';
        }
        if(cm!="\0")
        {
                hfile=_lopen(cm,OF_READWRITE);
                len=_llseek(hfile,0,2);
                _llseek(hfile,0,0);
                _lread(hfile,buf,len);
                _lclose(hfile);
                SetWindowText(hwnd,cm);
                SetWindowText(he,(LPSTR)buf);
                flag=FALSE;
        }

         maxx=GetSystemMetrics(SM_CXSCREEN);
         maxy=GetSystemMetrics(SM_CYSCREEN);
         hdc=GetDC(hwnd);
         memDC=CreateCompatibleDC(hdc);
         hBit=CreateCompatibleBitmap(hdc,maxx,maxy);
         SelectObject(memDC,hBit);
         hBrush=(HBRUSH)GetStockObject(WHITE_BRUSH);
         SelectObject(memDC,hBrush);
         PatBlt(memDC,0,0,maxx,maxy,PATCOPY);
         GetTextMetrics(hdc,&tm);
         strcpy(str,"R UUU  Sure!!!!!  ");
         for(i=0;i<NUMLINES;i++)
         {
         TextOut(memDC,x,y,str,strlen(str));
         TextOut(hdc,x,y,str,strlen(str));
         y=y+tm.tmHeight+tm.tmExternalLeading;
}
         ReleaseDC(hwnd,hdc);

         break;

 case WM_SIZE:

         MoveWindow(he,0,0,LOWORD(lParam),HIWORD(lParam),
         TRUE);
         break;
```

```
case WM_DESTROY:
        MessageBox(hwnd,"R U SURE?","close",0);
        PostQuitMessage(0);
        break;
case WM_COMMAND:
        switch(wParam)
        {
        case  400:
                        x=y=0;
                        PrintInit(&printdlg,hwnd);
                        if(!PrintDlg(&printdlg))
                                break;

                        docinfo.cbSize=sizeof(DOCINFO);
                        docinfo.lpszDocName="PRINTING TEXT";
                        docinfo.lpszOutput=NULL;
                        docinfo.lpszDatatype=NULL;
                        docinfo.fwType=0;
                        GetTextMetrics(printdlg.hDC,&tm);
                        GetWindowText(he,buf,-1);
                        len=strlen(buf);
                        strcpy(str,"R U SURE!!!!");
                        StartDoc(printdlg.hDC,&docinfo);
                        for(copies=0;copies<printdlg.nCopies;copies++)
                        {
                                StartPage(printdlg.hDC);
                                for(i=0;i<NUMLINES;i++)
                                {
                                TextOut(printdlg.hDC,x,y,buf,len);
                                y=y+tm.tmHeight+tm.tmExternalLeading;

                                }
                                EndPage(printdlg.hDC);
                        }
                        EndDoc(printdlg.hDC);
                        DeleteDC(printdlg.hDC);
                        break;
                case 401:
                        MessageBox(hwnd,"HELP MENUS NOT
                        AVAILABLE","HELP",MB_OK);

                        break;
```

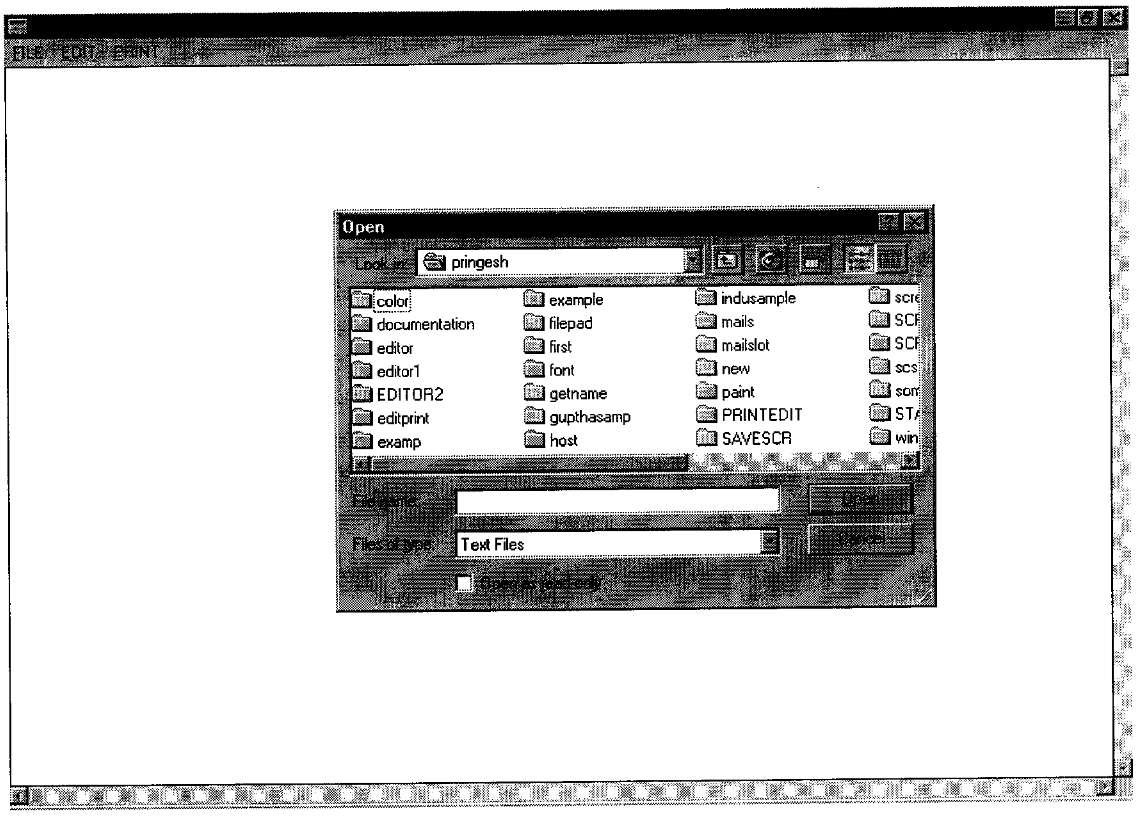
```
            case 1:
                    SetWindowText(hwnd,"Pringesh");
                    SetWindowText(he,"");
                    flag=TRUE;
                    break;
            case 2:
                    SetWindowText(he,"");
                    memset(&ofn,0,sizeof(OPENFILENAME));
                    ofn.lStructSize=sizeof(OPENFILENAME);
                    ofn.hwndOwner=hwnd;
                    ofn.lpstrFilter="Text Files\0*.txt\0All Files\0*.*\0\0";
                    ofn.lpstrFile=file;
                    ofn.nMaxFile=sizeof(file);
                    ofn.lpstrFileTitle=tit;
                    ofn.nMaxFileTitle=sizeof(tit);
                    ofn.lpstrDefExt="txt";
                    GetOpenFileName(&ofn);
                    if(!(hfile=_lopen(ofn.lpstrFile,OF_READWRITE)))
                    MessageBox(hwnd,"error","error",0);
                    len=_llseek(hfile,0,2);
                    _llseek(hfile,0,0);
                    _lread(hfile,buf,len);
                    _lclose(hfile);
                    SetWindowText(hwnd,ofn.lpstrFileTitle);
                    SetWindowText(he,(LPSTR)buf);
                    flag=false;
                    break;
            case 3:
                    if(flag!=TRUE)
                    {
                    GetWindowText(he,buf,-1);
                    len=strlen(buf);
                    MessageBox(hwnd,buf,"PREVIEW",0);
                    hfile=_lcreat(ofn.lpstrFile,0);
                    _lwrite(hfile,buf,len);
                    _lclose(hfile);
                    }
                    else goto label;
                    break;
label:
            case 4:
```

```
                        memset(&ofn,0,sizeof(OPENFILENAME));
                        ofn.lStructSize=sizeof(OPENFILENAME);
                        ofn.hwndOwner=hwnd;
                        ofn.lpstrFilter="Text Files\0*.txt\0All Files\0*.*\0\0";
                        ofn.lpstrFile=file;
                        ofn.nMaxFile=sizeof(file);
                        ofn.lpstrFileTitle=tit;
                        ofn.nMaxFileTitle=sizeof(tit);
                        ofn.lpstrDefExt="txt";
                        GetSaveFileName(&ofn);
                        if(!(hfile=_lcreat(ofn.lpstrFile,0)))
                        MessageBox(hwnd,"error","error",0);
                        GetWindowText(he,buf,-1);
                        len=strlen(buf);
                        _lwrite(hfile,buf,len);
                        _lclose(hfile);
                        SetWindowText(hwnd,ofn.lpstrFileTitle);
                        flag=FALSE;
                        break;
                case 30:
                        response=MessageBox(hwnd,"QUIT THE
                        PROGRAM?","EXIT",MB_YESNO);
                        if(response==IDYES)
                        PostQuitMessage(0);
                        break;
                case 11:
                        SendMessage(he,WM_CUT,0,0);
                        break;
                case 12:
                        SendMessage(he,WM_COPY,0,0);
                        break;
                case 13:
                        SendMessage(he,WM_PASTE,0,0);
                        break;
                case 14:
                        SendMessage(he,WM_CLEAR,0,0);
                        break;
                }
        default:
                return(DefWindowProc(hwnd,message,wParam,lParam));
        }
        return 0L;
}

BOOL CALLBACK DlgProc(HWND h,UINT m,WPARAM w,LPARAM l)
```

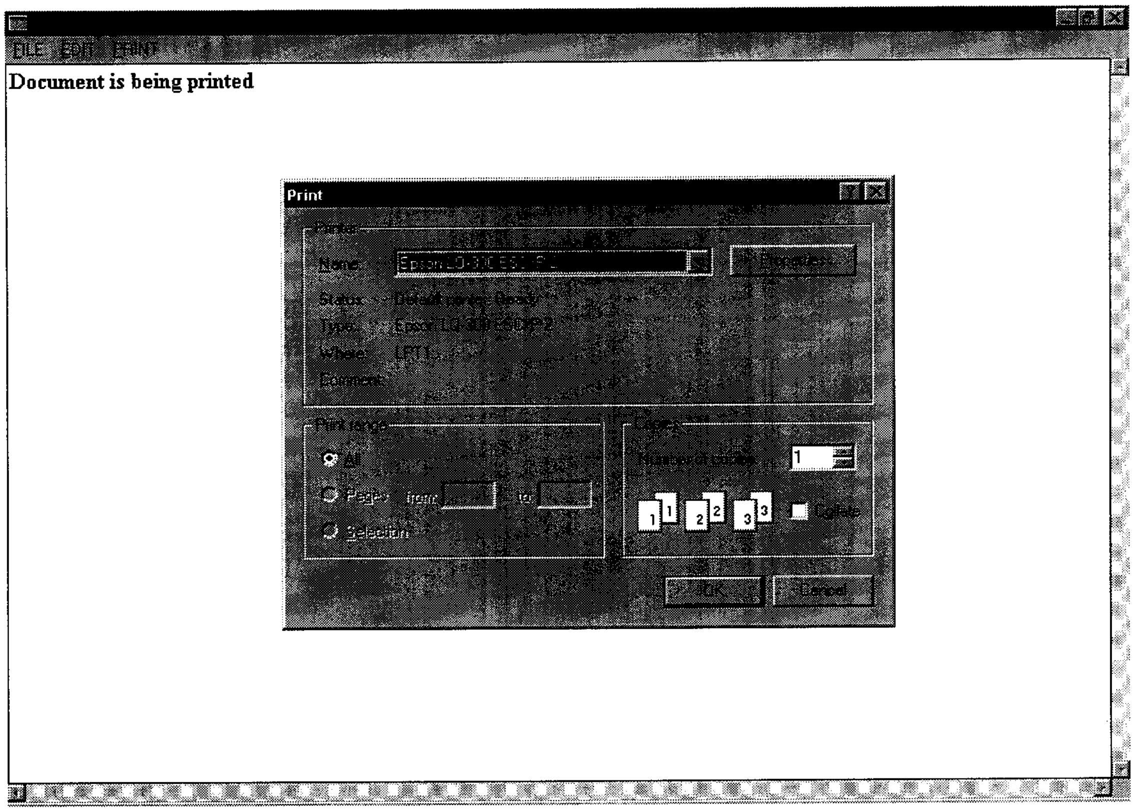
```
{
        switch(m)
        {
        case WM_COMMAND:
                switch(w)
                {
                        case 100:
                                break;
                }
                EndDialog(h,NULL);
                return TRUE;
        }
        return FALSE;
}

void PrintInit(PRINTDLG *printdlg,HWND hwnd)
{
printdlg->lStructSize =sizeof(PRINTDLG);
printdlg->hwndOwner =hwnd;
printdlg->hDevMode =NULL;
printdlg->hDevNames =NULL;
printdlg->hDC =NULL;

printdlg->Flags =PD_RETURNDC | PD_NOSELECTION |
                PD_NOPAGENUMS | PD_HIDEPRINTTOFILE;
printdlg->nFromPage =0;
printdlg->nToPage =0;
printdlg->nMinPage =0;
printdlg->nMaxPage =0;
printdlg->nCopies =1;
printdlg->hInstance =NULL;
printdlg->lCustData =0;
printdlg->lpfnPrintHook =NULL;
printdlg->lpfnSetupHook =NULL;
printdlg->lpPrintTemplateName =NULL;
printdlg->lpSetupTemplateName =NULL;
printdlg->hPrintTemplate =NULL;
printdlg->hSetupTemplate =NULL;
}
```

```
/*    Screen Saver    */


#include<windows.h>
#include<scrnsave.h>

char str[80]="NT 4 screen saver";
int delay=200;

LRESULT WINAPI ScreenSaverProc(HWND hwnd,UINT message,WPARAM
wParam,LPARAM lParam)

{
        static HDC hdc;
        static unsigned int timer;
        static RECT scrdim;
        static SIZE size;
        static int X=0,Y=0;
        static HBRUSH hBlkBrush;
        static TEXTMETRIC tm;

        switch (message)
        {


        case WM_CREATE:
                timer = SetTimer(hwnd,1,delay,NULL);
                hBlkBrush =(HBRUSH)GetStockObject(BLACK_BRUSH);
                break;

        case WM_ERASEBKGND:
                hdc=GetDC(hwnd);
                GetClientRect(hwnd,&scrdim);
                SelectObject(hdc,hBlkBrush);
                PatBlt(hdc,0,0,scrdim.right,scrdim.bottom,PATCOPY);
                GetTextMetrics(hdc,&tm);
                GetTextExtentPoint32(hdc,str,strlen(str),&size);
                ReleaseDC(hwnd,hdc);
                break;
        case WM_TIMER:
                hdc = GetDC(hwnd);
                SelectObject(hdc,hBlkBrush);
                PatBlt(hdc,X,Y,X+size.cx,Y+size.cy,PATCOPY);
                 X +=10; Y +=10;
                if(X>scrdim.right) X = 0;
                if(Y>scrdim.bottom) Y = 0;
                SetBkColor(hdc,RGB(0,0,0));
                SetTextColor(hdc,RGB(0,255,255));
                TextOut(hdc,X,Y,str,strlen(str));
                ReleaseDC(hwnd,hdc);
                break;


        case WM_DESTROY:
                KillTimer(hwnd,timer);
                break;


        default:
                return DefScreenSaverProc(hwnd,message,wParam,lParam);

        }

        return 0;

}

BOOL WINAPI ScreenSaverConfigureDialog (HWND hdwnd, UINT
message, WPARAM wParam, LPARAM lParam)
{

        return 0;

}

BOOL WINAPI RegisterDialogClasses(HINSTANCE hInst)
{

        return 1;

}
```

4.3.1 OPEN SCREEN



4.3.2 PRINT SCREEN



Document is being printed

Name: Epson LQ-300 ESC/P 2

Status: Default printer; Ready

Type: Epson LQ-300 ESC/P 2

Where: LPT1:

Comment:

[✓ All]

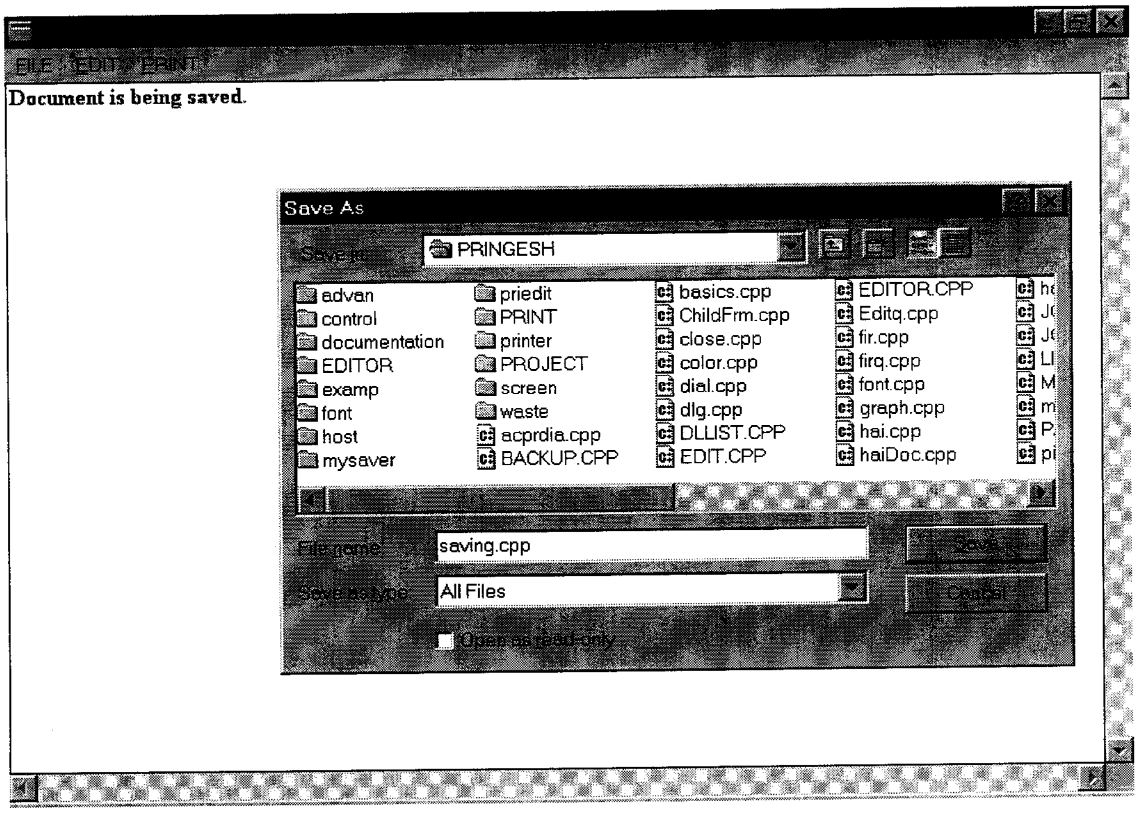
Pages from to

Selection

Number of copies: 1

Collate

4.3.3 SAVE SCREEN



Document is being saved.

PRINGESH

advan, control, documentation, EDITOR, examp, font, host, mysaver, priedit, PRINT, printer, PROJECT, screen, waste, acprdia.cpp, BACKUP.CPP, basics.cpp, ChildFrm.cpp, close.cpp, color.cpp, dial.cpp, dlg.cpp, DLLIST.CPP, EDIT.CPP, EDITOR.CPP, Editq.cpp, fir.cpp, firq.cpp, font.cpp, graph.cpp, hai.cpp, haiDoc.cpp

File name: saving.cpp

Save as type: All Files

5. SYSTEM TESTING AND IMPLEMENTATION

5.1. System Testing

Testing is done for the various test cases that were developed in order to catch all the problems and exceptions that arise during the real-time implementation of the system. The test cases were developed for unit testing as well as module testing.

5.1.1. Unit Testing

In this testing, individual components are tested to ensure that they operate correctly. Each component is tested independently, without the interference of other system components. With respect to this project, the individual functions in the DLL are treated as components and were tested. Also, the various activities are tested individually.

5.1.2. Module Testing

A module is a collection of dependent components such as collection of procedures and functions module encapsulates related components so it can be tested with other system components. Each of the activities, file management, printing, screen savers, editing modules and are segregated on the operations with the device. Thus each module is tested in this stage.

5.2. System Implementation

Implementation is one of the important stages in software development. In this application, the users are educated prior to using the system. The details of these services are explained about the overall flow in the system and are explained about the various events and exceptions that arise during a real time operation. Since this is a system application, it is to be added to the existing system services in order to completely make use of the services by all the users. These services are utilized only by the system in which the application is executing. The guidelines are provided in the help menu when the problem arose.

6. CONCLUSION

In this project we had developed Extended Services for Network Operating System. The extended services are editor, File Management, Screen Saver and Device Operations.

The editor is one in which a new document can be created, saved and can be altered to the required format. Any existing document can be viewed which is present in the remote node and it can also be modified.

In screen saver application, bitmap files are converted into screensaver files and those files can be customized.

File Management deals with the operations like creating directory, deleting it, modifying any file etc., irrespective of its location in the network and it is also possible to access a file from a remote node.

In device operation application, any device particularly printer, connected to any system in the network can be controlled. It is possible to print any remote file using a printer connected to a remote node.

7. BIBLIOGRAPHY

1. Brent E. Rector & Joseph M. Newcomer ., ***Win32 Programming***, Addison Wesley Longman Inc. , 1999.

2. Alok K. Sinha ., ***Network Programming in Windows NT,*** Addison Wesley Longman Inc. , 1999.

3. Raj Rajagopal & Subodh P. Monica ., ***Advanced Programming in Windows NT 4*** , Tata McGraw-Hill Companies Inc., *1998.*

4. Herbert Schildt ., ***Windows Programming*** , Tata McGraw-Hill Companies Inc., 1998.

5. Charles Petzold ., ***Windows Programming*** , Tata McGraw-Hill Companies Inc., 1998.

6. **MSDN** Online Help

   www.msdn.com
   www.iosoftware.com
   www.bioapi.com

8. APPENDIX

8.1 A Windows NT Skeleton

All Windows NT programs have certain things in common. In this section a skeleton is developed that provides these necessary features.

A minimal Windows NT program contains two functions: WinMain() and the window function. The WinMain() function must perform the following general steps:

1. Define a window class.
2. Register that class with Windows NT.
3. Create a window of that class.
4. Display the window.
5. Begin running the message loop.

Using the following program, minimal Windows NT skeleton is illustrated below.

```
/* A minimal Windows NT Skeleton  */

 #include<windows.h>

LRESULT CALLBACK WindowFunc (HWND, UINT, WPARAM, LPARAM);

char szWinName[]= “MyWin”;  /* name of window class */

int      WINAPI      WinMain(HINSTANCE      hThisInst,HINSTANCE
hPrevInst,LPSTR lpszArgs,int nWinMode)
{
```

```
WNDCLASSEX w;
MSG msg;
HWND hwnd;

/* Define a WindowClass */
w.cbSize = sizeof(WNDCLASSEX);

w.hInstance=hThisInst;          /* handle to this instance */
w.lpszClassName=szWinName;  /* window class name */
w.lpfnWndProc=WindowFunc;   /* window function */
w.style=0                        /* default style */

w.hIcon=LoadIcon(NULL,IDI_APPLICATION); /* standard icon */
w.hIconSm=LoadIcon(NULL,IDI_WINLOGO); /* standard icon */
w.hCursor=LoadCursor(NULL,IDC_ARROW);/ * cursor style */
w.lpszMenuName=NULL;                        / * no menu */
w.cbClsExtra=0;                                /* no extra */
w.cbWndExtra=0;                              /* information needed */

/* Make the window background white. */
w.hbrBackground=(HBRUSH)GetStockObject(0);

/* Register the window class. */
if (!RegisterClassEx(&w)) return 0;

/* Now that a window class has been registered, a window can be
created. */
```

```
    hwnd = CreateWindow(
    szWinName,                /* name of window class */
    "caption",                /* Title */
    WS_OVERLAPPEDWINDOW, /* window style - normal */
    0,                        /* X coordinate */
    0,                        /* Y coordinate */
    500,                      /* Width */
    500,                      /* Height */
    NULL,                     /* No parent window */
    NULL,
    hThisInst, /* handle of this instance of the program */
    NULL      /* no additional arguments */
    );

    /* Display the window */
    ShowWindow (hwnd,nWinMode);
    UpdateWindow (hwnd);

    /* Create the message loop */
   while(GetMessage(&msg,NULL,0,0))
    {
     TranslateMesage (&msg);   /* allow use of keyboard */
     DispatchMessage (&msg); /*return control to Windows NT */
    }
   return msg.wParam;
}

    /* This function is called by Windows NT and is passed
```

```
        messages from the message queue.*/
LRESULT        CALLBACK        WindowFunc(HWND        hwnd,UINT
message,WPARAM wParam,LPARAM lParam)
{
        switch(message)
        {
                case WM_DESTROY:  /* terminate the program */
                        PostQuitMessage(0);
                        break;
                default :
                /* Let Windows NT process any messages not specified in
                the preceding switch statement.  */

                return DefWindowProc(hwnd,message,wparam,lparam);
        }
        return 0;
}
```

This program is the basis for all the windows based applications. This can be enhanced to develop a more powerful Windows NT application.

8.2 API Functions

_lopen

The **_lopen** function opens an existing file and sets the file pointer to the beginning of the file.

```
HFILE _lopen (
  LPCSTR lpPathName,  // pointer to name of file to open
  int iReadWrite      // file access mode
);
```

Parameters

lpPathName

Pointer to a null-terminated string that names the file to open.

iReadWrite

Specifies the modes in which to open the file. This parameter consists of one access mode and an optional share mode.

Value	**Meaning**
OF_READ	Opens the file for reading only.
OF_READWRITE	Opens the file for reading and writing.
OF_WRITE	Opens the file for writing only.

_llseek

The **_llseek** function repositions the file pointer in a previously opened file.

```
LONG _llseek (
  HFILE hFile,  // handle to file
  LONG lOffset,   // number of bytes to move
  int iOrigin        // position to move from
);
```

Parameters

hFile

Handle to the file.

lOffset

Specifies the number of bytes the file pointer is to be moved.

iOrigin

Specifies the starting position and direction of the file pointer.

TextOut

The **TextOut** function writes a character string at the specified location, using the currently selected font, background color, and text color.

```
BOOL TextOut (
  HDC hdc,                // handle to device context
  int nXStart,            // x-coordinate of starting position
  int nYStart,            // y-coordinate of starting position
  LPCTSTR lpString,       // pointer to string
  int cbString            // number of characters in string
);
```

Parameters

hdc

Handle to the device context.

nXStart

Specifies the logical x-coordinate of the reference point that the system uses to align the string.

nYStart

Specifies the logical y-coordinate of the reference point that the system uses to align the string.

lpString

Pointer to the string to be drawn. The string does not need to be zero-terminated, since *cbString* specifies the length of the string.

cbString

Specifies the number of characters in the string.

PrintDlg

The **PrintDlg** function displays a **Print** dialog box or a **Print Setup** dialog box.

BOOL PrintDlg (

LPPRINTDLG *lppd* // pointer to structure with initialization data

);

Parameters

lppd

Pointer to a ***PRINTDLG structure*** that contains information used to initialize the dialog box.

memset

Sets buffers to a specified character.

void *memset(void ****dest*, **int *c*, **size_t** *count* **);**

Parameters

dest

Pointer to destination

c

Character to set

count

Number of characters

GetHostName

Syntax

public *String* getHostName()

Returns

The host name for this IP address.

Description

Returns the hostname for this address.

gethostbyname

The Windows Sockets **gethostbyname** function retrieves host information corresponding to a host name from a host database.

struct hostent FAR * gethostbyname (

const char FAR * *name*

);

Parameters

name

A pointer to the null-terminated name of the host to resolve.

Getservbyname

The Windows Sockets **getservbyname** function retrieves service information corresponding to a service name and protocol.

struct servent FAR * getservbyname (

  const char FAR * *name*,

  const char FAR * *proto*

);

Parameters

name

[in] A pointer to a null-terminated service name.

proto

[in] An optional pointer to a null-terminated protocol name.

bind

The Windows Sockets **bind** function associates a local address with a socket.

int bind (

  SOCKET *s*,

  const struct sockaddr FAR* *name*,

  int *namelen*

);

Parameters

s

[in] A descriptor identifying an unbound socket.

name

[in] The address to assign to the socket from the ***SOCKADDR*** structure.

namelen

[in] The length of the *name*.

setsockopt

The Windows Sockets **setsockopt** function sets a socket option.

int setsockopt (

  SOCKET *s*,

  int *level*,

  int *optname*,

  const char FAR * *optval*,

  int *optlen*

);

Parameters

s

[in] A descriptor identifying a socket.

level

[in] The level at which the option is defined; the supported *level*s include SOL_SOCKET and IPPROTO_TCP.

optname

[in] The socket option for which the value is to be set.

optval

[in] A pointer to the buffer in which the value for the requested option is supplied.

optlen

[in] The size of the *optval* buffer.

CreateWindow

The **CreateWindow** function creates an overlapped, pop-up, or child window. It specifies the window class, window title, window style, and the initial position and size of the window.

```
HWND CreateWindow (
  LPCTSTR lpClassName,      // pointer to registered class name
  LPCTSTR lpWindowName,     // pointer to window name
  DWORD dwStyle,            // window style
  int x,                    // horizontal position of window
  int y,                    // vertical position of window
  int nWidth,               // window width
  int nHeight,              // window height
  HWND hWndParent,          // handle to parent or owner window
  HMENU hMenu,              // handle to menu or child-window
                            // identifier
  HANDLE hInstance,         // handle to application instance
  LPVOID lpParam            // pointer to window-creation data
);
```

Parameters

lpClassName

Pointer to a null-terminated string . If *lpClassName* is a string, it specifies the window class name.

lpWindowName

Pointer to a null-terminated string that specifies the window name.

dwStyle

Specifies the style of the window being created.

Style	**Meaning**
WS_HSCROLL	Creates a window that has a horizontal scroll bar.
WS_MAXIMIZE	Creates a window that is initially maximized.
WS_OVERLAPPEDWINDOW	Creates an overlapped window
WS_VISIBLE	Creates a window that is initially maximized.
WS_VSCROLL	Creates a window that has a vertical maximized.

x

Specifies the initial horizontal position of the window.

y

Specifies the initial vertical position of the window.

nWidth

Specifies the width, in device units, of the window.

nHeight

Specifies the height, in device units, of the window.

hWndParent

Handle to the parent or owner window of the window being created.

hMenu

Handle to a menu.

DispatchMessage

The **DispatchMessage** function dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the ***GetMessage*_** function.

```
LONG DispatchMessage (
  CONST MSG *lpmsg   // pointer to structure with message
);
```

Parameters

lpmsg

Pointer to a ***MSG*_**structure that contains the message.

PostQuitMessage

The **PostQuitMessage** function indicates to the system that a thread has made a request to terminate (quit). It is typically used in response to a ***WM_DESTROY*** message.

```
VOID PostQuitMessage (
  int nExitCode   // exit code
);
```

Parameters

nExitCode

Specifies an application exit code. This value is used as the *wParam* parameter of the ***WM_QUIT*** message.