**OBJECTIVE & SCOPE OF THE PROJECT**

**Title and Objective of the Project**

The title of the project is ‘WinMonitor’. As the name suggests, it is a system utility project for monitoring the Windows based systems. It is an effective administrative tool, by which one can monitor all systems in a LAN.

**Tools/Platform used**

This is essentially a Windows based system project so using platforms such as Windows-NT based systems, Windows-98 systems are appropriate.

The tools used are Visual Basic.NET and Visual C++.NET, so that they ensures the COM (Component Object Model) and Customization. The software has its own API.

This project is a system utility which adopts the client/server architecture. Client and Server modules are being developed using different tools as mentioned below.

**Visual C++.NET:** Whole of the server components and the necessary reusable DLL’s are coded using VC++.NET, because of its power and flexibility. Almost all part of the network interface are developed using this language.

**Visual Basic.NET:** All the user interfaces are developed using this language, because of its power and simplicity. This language is easier to learn and use. It will help you to reduce the coding time required, using its large collection of reusable components. A graphical User interface requires a greater deal of coding, so I adopt this language for coping with the effort required.

**XML:** All the configuration files required by the client and server are intended to keep in XML documents because of its world wide acceptance. It is almost become a standard in the industry. Large arrays of tools are available for creating, maintaining and accessing XML documents under VC++.NET and VB.NET based software development.

**Modules covered in the Project**

The project is essentially a networked, system utility project whose components are distributed on various different systems. The project allows a System Administrator to monitor all systems under the network for which he has the charge without physically moving between hosts.

All machines other than the central server will contain a component called *‘System Monitor’* which will keep track of all activities happening in the system. It will either send this information to the System Administrator to his mail box or it will store it in the same system for future mailing.

The other component is called the *‘Administrator Unit’* which will run on the server system. It will include a *‘System Monitor’* building unit, and a Sender unit for sending this Monitor to individual systems in the network, and instructing them to send the system information to appropriate mailboxes at regular intervals of time. This module also includes a ‘On Line Host Monitor’ by which you can explore on a remote host online, a ‘Maintenance Monitor’ to reconfigure and maintains relationships with already acquired hosts. This latter one is also responsible for generating reports.

The *‘System Monitor’* will contain a Key Logger, a Screen Snatcher, a Process Information Snatcher, an Addins Monitor and a Mailing System to send the information so far collected with in the system to appropriate mailbox. It also contains a Storage Manager to store details so far collected if the network is not available. ‘On Line Request Monitor’ will handle online request for system information from a remote Administrator. A symbolic division is as follows;

**WinMonitor.1.0 Server Modules**

1. Remote file and directory explorer.
2. Remote file and directory copier.
3. Remote screen shot snatcher.
4. Remote system shutdown facility.
5. Remote system online and offline key logging.
6. Remote system process explorer.
   1. Process terminator.
   2. Process priority level changer
7. Compression and decompression using LZSS algorithm.
8. Remote chat facility.
9. Remote disk monitoring.
10. Remote memory monitoring.
11. Fake messaging system.

**WinMonitor.1.0 Client Modules**

1. Server builder with full options.
2. Server controller.
   1. Remote server explorer controller.
   2. Remote server process explorer controller.
   3. Remote server controller.
3. Server configuration XML builder.
4. EXE resource embedding facility.
5. Client GUI for controlling server.
6. Client network interface manager.
7. Connected and build servers list generator.

**Scope & Advantages**

As I already mentioned this is a system utility which can be used by an administrator to monitor all the systems under his LAN. He no longer goes to individual system for the sake of monitoring; rather he will sit in front his own machine and using WinMonitor, he will be able to control and monitor all the systems in the network.

**Disadvantages**

Since I had to complete this project within a short period of time some of the modules, which I supposed to do are eliminated. I already mentioned that the software is for monitoring all the machines in a LAN. If we have a good deal of time the software can be made perfectly adaptable for running in the Internet, so that we can access and monitor any intended system with in the global network. Since I have to complete the project within this short time span, I have to drop this module that I not necessarily intended.

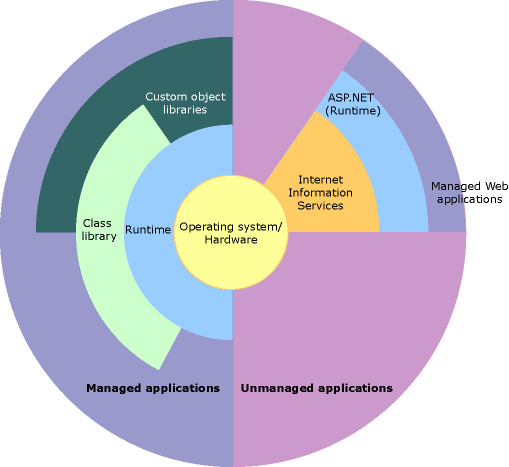
**TOOLS/PLATFORM IN WHICH THE PROJECT IS DEVELOPED**

This is essentially a Windows based system project so using platforms such as Windows-NT based systems, Windows-98 systems are appropriate. The tools used are Visual Basic.NET and Visual C++.NET, so that they ensures the COM (Component Object Model) and Customization. The software has its own native API.

Actually I used the CLIENT/SERVER architecture to develop the project. The CLIENT part is entirely developed using Visual Basic.NET and the server part uses Visual C++.NET (VC++.NET) because of its power and accessibility to low level Windows API’s in a simple way. Below I introduce a brief description of these tools.

**ABOUT .NET TECHNOLOGY**

**.NET Framework in context**



**.NET Framework Advantages**

The .NET Framework offers a number of advantages to developers. The following paragraphs describe them in detail.

**Consistent Programming Model**

Different programming languages have different approaches for doing a task. For example, accessing data with a VB 6.0 application and a VC++ application is totally different. When using different programming languages to do a task, a disparity exists among the approach developers use to perform the task. The difference in techniques comes from how different languages interact with the underlying system that applications rely on.

With .NET, for example, accessing data with a VB .NET and a C# .NET looks very similar apart from slight syntactical differences. Both the programs need to import the System.Data namespace, both the programs establish a connection with the database and both the programs run a query and display the data on a data grid. The VB 6.0 and VC++ example mentioned in the first paragraph explains that there is more than one way to do a particular task within the same language. The .NET example explains that there's a unified means of accomplishing the same task by using the .NET Class Library, a key component of the .NET Framework.

The functionality that the .NET Class Library provides is available to all .NET languages resulting in a consistent object model regardless of the programming language the developer uses.

**Direct Support for Security**

Developing an application that resides on a local machine and uses local resources is easy. In this scenario, security isn't an issue as all the resources are available and accessed locally. Consider an application that accesses data on a remote machine or has to perform a privileged task on behalf of a nonprivileged user. In this scenario security is much more important as the application is accessing data from a remote machine.

With .NET, the Framework enables the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. This makes distributed computing more secure because .NET developers cooperate with network security devices instead of working around their security limitations.

**Simplified Development Efforts**

Let's take a look at this with Web applications. With classic ASP, when a developer needs to present data from a database in a Web page, he is required to write the application logic (code) and presentation logic (design) in the same file. He was required to mix the ASP code with the HTML code to get the desired result.

ASP.NET and the .NET Framework simplify development by separating the application logic and presentation logic making it easier to maintain the code. You write the design code (presentation logic) and the actual code (application logic) separately eliminating the need to mix HTML code with ASP code. ASP.NET can also handle the details of maintaining the state of the controls, such as contents in a textbox, between calls to the same ASP.NET page.

Another advantage of creating applications is debugging. Visual Studio .NET and other third party providers provide several debugging tools that simplify application development. The .NET Framework simplifies debugging with support for Runtime diagnostics. Runtime diagnostics helps you to track down bugs and also helps you to determine how well an application performs. The .NET Framework provides three types of Runtime diagnostics: Event Logging, Performance Counters and Tracing.

**Easy Application Deployment and Maintenance**

The .NET Framework makes it easy to deploy applications. In the most common form, to install an application, all you need to do is copy the application along with the components it requires into a directory on the target computer. The .NET Framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET Framework ensures that all the components the application depends on are available on the computer before the application begins to execute.

**ABOUT VISUAL BASIC.NET**

**VB Language**

Visual Basic, the name makes me feel that it is something special. In the History of Computing world no other product sold more copies than Visual Basic did. Such is the importance of that language which clearly states how widely it is used for developing applications. Visual Basic is very popular for it's friendly working (graphical) environment. Visual Basic. NET is an extension of Visual Basic programming language with many new features in it. The changes from VB to VB .NET are huge, ranging from the change in syntax of the language to the types of projects we can create now and the way we design applications. Visual Basic .NET was designed to take advantage of the .NET Framework base classes and runtime environment. It comes with power packed features that simplify application development.

**Briefly on some changes:**

The biggest change from VB to VB .NET is, VB .NET is Object-Oriented now. VB .NET now supports all the key OOP features like Inheritance, Polymorphism, Abstraction and Encapsulation. We can now create classes and objects, derive classes from other classes and so on. The major advantage of OOP is code reusability

The way we handle data with databases is changed as well. VB .NET now uses ADO .NET, a new data-handling model to communicate with databases on local machines or on a network and also it makes handling of data on the Internet easy. All the data in ADO .NET is represented in XML format and is exchanged in the same format. Representing data in XML format allows us for sending large amounts of data on the Internet and it also reduces network traffic when communicating with the database VB .NET now supports Multithreading. A threaded application allows to do number of different things at once, running different execution threads allowing to use system resources  
Web Development is now an integral part of VB .NET making Web Forms and Web Services two major types of applications

**Namespaces**

A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace. The namespace with all the built-in VB functionality is the System namespace. All other namespaces are based on this System namespace.

**Assemblies**

An assembly is the building block of a .NET application. It is a self describing collection of code, resources, and metadata (data about data, example, name, size, version of a file is metadata about that file). An Assembly is a complied and versioned collection of code and metadata that forms an atomic functional unit. Assemblies take the form of a  dynamic link library (.dll) file or executable program file (.exe) but they differ as they contain the information found in a type library and the information about everything else needed to use an application or component. All .NET programs are constructed from these Assemblies. Assemblies are made of two parts: manifest, contains information about what is contained within the assembly and modules, internal files of IL code which are ready to run. When programming, we don't directly deal with assemblies as the CLR and the .NET framework takes care of that behind the scenes. The assembly file is visible in the Solution Explorer window of the project.

An assembly includes:

* Information for each public class or type used in the assembly – information includes class or type names, the classes from which an individual class is derived, etc
* Information on all public methods in each class, like, the method name and return values (if any)
* Information on every public parameter for each method like the parameter's name and type
* Information on public enumerations including names and values
* Information on the assembly version (each assembly has a specific version number)
* Intermediate language code to execute
* A list of types exposed by the assembly and list of other assemblies required by the assembly

**.NET Defined**

Before getting deeply into the subject we will first know how Businesses are related to Internet, what .NET means to them and what exactly .NET is built upon. As per the product documentation from a Business perspective, there are three phases of the Internet. The First phase gets back to the early 1990's when Internet first came into general use and which brought a big revolution for Businesses. In the First phase of the Internet Businesses designed and launched their Website's and focused on the number of hits to know how many customers were visiting their site and interested in their products, etc. The Second phase is what we are in right now and in this phase Businesses are generating revenue through Online Transactions. We are now moving into the Third phase of the Internet where profit is the main priority. The focus here is to Businesses effectively communicate with their customers and partners who are geographically isolated, participate in Digital Economy and deliver a wide range of services. How can that be possible? The answer, with .NET.

**What is .NET?**

Many people reckon that it's Microsoft's way of controlling the Internet, which is false. .NET is Microsoft's strategy of software that provides services to people any time, any place, on any device. An accurate definition of .NET is, it's an XML Web Services  platform which allows us to build rich .NET applications, which allows users to interact with the Internet using wide range of smart devices (tablet devices, pocket PC's, web phones etc), which allows to build and integrate Web Services and which comes with many rich set of tools like Visual Studio to fully develop and build those applications.

**What are Web Services?**

Web Services are the applications that run on a Web Server and communicate with other applications. It uses a series of protocols to respond to different requests. The protocols on which Web Services are built are summarized below:

**UDDI**: Stands for Universal Discovery and Description Integration. It's said to be the Yellow Pages of Web Services which allows Businesses to search for other Businesses allowing them to  search for the services it needs, know about the services and contact them.

**WSDL**: Stands for Web Services Description Language, often called as whiz-dull. WSDL is an XML document that describes a set of SOAP messages and how those messages are exchanged.

**SOAP**: Stands for Simple Object Access Protocol. It's the communication protocol for Web Services.

**XML**, **HTTP** and **SMTP**: Stands for Extensible Markup Language, Hyper Text Transfer Protocol and Simple Message Transfer Protocol respectively. UDDI, WSDL and SOAP rely on these protocols for communication.

The image below shows the order of the protocols on which Web Services are built:



**Example of a Web Services Application**

Let's say a customer accesses a Website and buys something. The Web services of the business will communicate with the inventory system to see if there is enough stock to fulfill the order. If not, the system can communicate with the suppliers to find one or all of the parts that make up the order before filling the order. At all stages the customer will be kept informed via messages. The end result is a seamless system communicating and exchanging information easily regardless of the platform they are all running on. The business don't need to worry about going to the wrong supplier because it asks the Web service running on the supplier system what it does. And the business doesn't have to worry about the other system's methods of handling data because they communicate via SOAP and XML.

**Real World Application:**

Microsoft's passport service is an example of a .NET service. Passport is a Web-based service designed to make signing in to Websites fast and easy. Passport enables participating sites to authenticate a user with a single set of sign-in credentials eliminating the need for users to remember numerous passwords and sign-in names. You can use one name and password to sign in to all .NET Passport-participating sites and services. You can store personal information in your .NET Passport profile and, if you choose, automatically share that information when you sign in so that participating sites can provide you with personalized services. If you use Hotmail for your email needs then you should be very much familiar with the passport service.

To find out more about how Businesses are implementing Web Services and the advantages it is providing please visit Microsoft's Website and check out the case studies published.

**What is .NET Built On?**

.NET is built on the Windows Server System to take major advantage of the OS and which comes with a host of different servers which allows for building, deploying, managing and maintaining Web-based solutions. The Windows Server System is designed with performance as priority and it provides scalability, reliability, and manageability for the global, Web-enabled enterprise. The Windows Server System integrated software products are built for interoperability using open Web standards such as XML and SOAP.

**.NET and XML**

There is a lot of connection between XML and .NET. XML is the glue that holds .NET together. XML looks similar to HTML which  is readable and text-based. XML is a method of putting structured data into a text file. XML is the specification for defining the structure of the document. Around this specification a whole family of optional modules are being developed. The reason why XML is linked so much to .NET is, it's platform independent and is well supported on any environment. To move the data contained in an XML file around different organizations using different software on different platforms it should be packed it into something. That something is a protocol like SOAP.

**About SOAP**

SOAP, Simple Object Access Protocol is a simple, lightweight protocol for exchanging information between peers in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelop that describes what is in the message and how it should be processed, a set of encoding rules and a convention for representing remote procedure calls and responses.

**.NET vs Java**

Many of us wonder what .NET has to do with Java. Is there any relation between them? Are they similar? and so on. I even hear some people say .NET is Microsoft's answer to Java. I think every language has its own pros and cons. Java is one of the greatest programming languages created by humans. Java doesn't have a visual interface and requires us to write heaps of code to develop applications. On the other hand, with .NET, the Framework supports around 20 different programming languages which are better and focus only on business logic leaving all other aspects to the Framework. Visual Studio .NET comes with a rich visual interface and supports drag and drop. Many applications were developed, tested and maintained to compare the differences between .NET and Java and the end result was a particular application developed using .NET requires less lines of code, less time to develop and lower deployment costs along with other important issues. Personally, I don't mean to say that Java is gone or .NET based applications are going to dominate the Internet but I think .NET definitely has an extra edge as it is packed with features that simplify application development.

I hope the information above puts some light on the technology aspects behind .NET and helps you in getting started.

**ABOUT VISUAL C++.NET**

**Why I Chose C++**

A good way to get into an argument with a computer programmer is to attempt to explain why the language they are using is not as good as the one you are using. Most of the programmers I know are positively religious over their Operating System, their development language and finally their text editor.

Once again, classes are one of the three most important things in C++. They are often used to store data useful for a specific part in your program. For instance: suppose there is a BIG company you run, this company has information about every employee working there. This information could be stored in a class named employee, a single item of this information is called a member of employee. This is basically what classes do here, store information on a particular group (at least in this example). But a class can do more, a class can contain member functions which are functions that operate within a class that alter or do something with the members of the class.

As you might have inferred by the title, I feel that C++ is the superior computer programming language. I will begin by qualifying that statement somewhat. I learned to program using Pascal and I still feel that it is a good language for learning computer programming. Pascal is type safe, has a very limited number of keywords and encourages good programming principles. The BASIC language in the form of Visual Basic on the other hand is an ideal language for quickly putting together a project, and for taking advantage of data base access and other advanced programming tools such as Microsoft's Component Object Model (COM).

Why is it then that modern operating systems and very large applications are written in C++? Visual Basic and Pascal (in the form of Delphi) barely resemble the ANSI definition of these languages. Each of these languages are proprietary and lock you in to a particular vendor. This does not particularly bother me, since I programmed exclusively for the Windows environment and use Microsoft's Visual C++ as my development environment. There are other features of C++ that are attractive when doing medium to large-scale development:

* Operator overloading (also in Delphi)
* Exceptions (also in Delphi and Java)
* Templates

These are the primary features of C++ that set it apart from its peers. I am not going to mention Object Oriented Programming (OOP) here because Java, VB and Delphi support OOP in some form or fashion. I would place OOP above the other three bullets if I were including it in this discussion.

**The Callback Function**

Unlike stand-alone MS-DOS mode applications that did not have to share system resources, Windows applications must coexist and behave themselves in relation to any previously loaded Windows programs. For example, your computer has only one keyboard. If you have three programs loaded, how would any one application decide which one receives keyboard input. The answer is that each application could not do this. The only way this process can work effectively is if the operating system oversees each application’s needs. For this process to take place, every Windows application must create what is known as a

*Callback* function.

The purpose of the callback function is to report to Windows, via messages, what the application wants to do, while Windows informs each application, again via messages, what’s out there of interest, i.e., current mouse coordinates. Let’s take a detailed look at this statement in swt.cpp (the actual **WndProc()** function body is explained in detail later on in this chapter). Every C/C++ function prototype begins with the function’s return type, in this case LRESULT CALLBACK. LRESULT defines the type used for the return value of window procedures. It is a 32-bit value that can at times be broken down into two 16-bit values called *low* and *high*.

CALLBACK identifies the function as being an application-defined function that a system or subsystem (Windows, for example) calls. Typically, this happens when an event occurs or when windows or fonts are being enumer-The function name WndProc() is not a reserved identifier. Any legal function name is allowed:

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

Understanding the Simple Windows Template

**A Brief Word about Handle*s***

Writing a Windows application always involves the use of handles. A *Handle* is a unique number that identifies many different types of objects, such as windows, controls, menus, icons, pens and brushes, memory allocation, output devices, and even window instances. In Windows terminology, each loaded copy of a program is called an *instance*.

Because Windows allows you to run more than one copy of the sameapplication at the same time, it needs to keep track of each of these instances. It does this by attaching a unique instance handle to each running copy of the application. C/C++ code style conventions use all uppercase identifiers to define nonstandard C/C++ data types. To find the definitions for these uppercase identifiers, you need to go to their defining header files referenced via windows.h. Usually, the instance handle is used as an index into an internally maintained table. Having the handle reference a table element rather than an actual memory address allows Windows to rearrange all resources dynamically by simply inserting a new address into the resource’s identical table position. For

example, if Windows associates a particular application’s resource with table look-up position 3, then no matter where Windows moves the resource in memory, table position 3 will contain the resource’s current location. Windows conserves memory resources by the way it manages multiple instances of the same application. Several multitasking environments load each duplicate instance of an application, just as if it were an entirely new application. However, Windows can conserve system resources by using the same code for all instances of an application. The only portion of each instance that is usually unique is the instance’s data segment. The first instance of an application has a very important role. It is the first instance of an application that defines all of the objects necessary for the functioning of the application. This can include controls, menus, dialog boxes, and much more, along with new window classes. A Windows application can even be instructed to allow other applications to share these new definitions. The callback function prototype is followed by a simple program name string definition. In case you have never heard of Hungarian notation, the *sz* in front of *ProgName[]* represents an abbreviation for the variable’s data type, in this case, a string (*s*) that is null-terminated (*z* = ’\0’). char szProgName[]=”ProgName”;

**The ‘WinMain’ function**

“int WINAPI WinMain

(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow)”

WinMain() is windows equivalent of main() from DOS or UNIX. This is where your program starts execution. The parameters are as follows:

HINSTANCE hInstance

Handle to the programs executable module (the .exe file in memory)

HINSTANCE hPrevInstance

Always NULL for Win32 programs.

LPSTR lpCmdLine

The command line arguments as a single string. NOT including the program name.

int nCmdShow

An integer value which may be passed to ShowWindow(). We'll get to this later.

hInstance is used for things like loading resources and any other task which is performed on a per-module basis. A module is either the EXE or a DLL loaded into your program. For most (if not all) of this tutorial, there will only be one module to worry about, the EXE.

hPrevInstance used to be the handle to the previously run instance of your program (if any) in Win16. This no longer applies. In Win32 you ignore this parameter.

**Calling Conventions**

WINAPI specifies the calling convention and is defined as \_stdcall. If you don't know what this means, don't worry about it as it will not really affect us for the scope of this tutorial. Just remember that it's needed here.

**Win32 Data Types**

You will find that many of the normal keywords or types have windows specific definitions, UINT for unsigned int, LPSTR for char\* etc... Which you choose is really up to you. If you are more comfortable using char\* instead of LPSTR, feel free to do so. Just make sure that you know what a type is before you substitute something else.

Just remember a few things and they will be easy to interpret. An LP prefix stands for *Long Pointer*. In Win32 the *Long* part is obsolete so don't worry about it. And if you don't know what a pointer is, you can either 1) Go find a book or tutorial on C, or 2) just go ahead anyway and screw up a lot. I'd really recommend #1, but most people go with #2 (I would :). But don't say I didn't warn you.

Next thing is a C following a LP indicates a const pointer. LPCSTR indicates a pointer to a const string, one that can not or will not be modified. LPSTR on the other hand is not const and may be changed.

You might also see a T mixed in there. Don't worry about this for now, unless you are intentionally working with *Unicode*, it means nothing.

**Understanding the Message Loop**

Understanding the message loop and entire message sending structure of windows programs is essential in order to write anything but the most trivial programs. Now that we've tried out message handling a little, we should look a little deeper into the whole process, as things can get very confusing later on if you don't understand why things happen the way they do.

**What is a Message?**

A message is an integer value. If you look up in your header files (which is good and common practice when investigating the workings of API's) you can find things like:

#define WM\_INITDIALOG 0x0110

#define WM\_COMMAND 0x0111

#define WM\_LBUTTONDOWN 0x0201

...and so on. Messages are used to communicate pretty much everything in windows at least on basic levels. If you want a window or control (which is just a specialized window) to do something you send it a message. If another window wants you to do something it sends you a message. If an event happens such as the user typing on the keyboard, moving the mouse, clicking a button, then messages are sent by the system to the windows affected. If you are one of those windows, you handle the message and act accordingly.

Each windows message may have up to two parameters, wParam and lParam. Originally wParam was 16 bit and lParam was 32 bit, but in Win32 they are both 32 bit. Not every message uses these parameters, and each message uses them differently. For example the WM\_CLOSE message doesn't use either, and you should ignore them both. The WM\_COMMAND message uses both, wParam contains *two* values, HIWORD(wParam) is the notification message (if applicable) and LOWORD(wParam) is the control or menu id that sent the message. lParam is the HWND (window handle) to the control which sent the message or NULL if the messages isn't from a control.

HIWORD() and LOWORD() are macros defined by windows that single out the two high bytes (High Word) of a 32 bit value (0x**FFFF**0000) and the low word (0x0000**FFFF**) respectively. In Win32 a WORD is a 16bit value, making DWORD (or Double Word) a 32bit value.

To send a message you can use PostMessage() or SendMessage(). PostMessage() puts the message into the *Message Queue* and returns immediatly. That means once the call to PostMessage() is done the message may or may not have been processed yet. SendMessage() sends the message directly to the window and does not return untill the window has finished processing it. If we wanted to close a window we could send it a WM\_CLOSE message like this PostMessage(hwnd, WM\_CLOSE, 0, 0); which would have the same effect as clicking on the close button on the top of the window. Notice that wParam and lParam are both 0. This is because, as mentioned, they aren't used for WM\_CLOSE.

**Dialogs**

Once you begin to use dialog boxes, you will need to send messages to the controls in order to communicate with them. You can do this either by using GetDlgItem() first to get the handle to the control using the ID and then use SendMessage(), OR you can use SendDlgItemMessage() which combines the steps. You give it a window handle and a child ID and it will get the child handle, and then send it the message. SendDlgItemMessage() and similar APIs like GetDlgItemText() will work on all windows, not just dialog boxes.

**What is the Message Queue**

Lets say you were busy handling the WM\_PAINT message and suddenly the user types a bunch of stuff on the keyboard. What should happen? Should you be interrupted in your drawing to handle the keys or should the keys just be discarded? Wrong! Obviously neither of these options is reasonable, so we have the message queue, when messages are posted they are added to the message queue and when you handle them they are removed. This ensure that you aren't going to miss messages, if you are handling one, the others will be queued up untill you get to them.

**What is a Message Loop**

while(GetMessage(&Msg, NULL, 0, 0) > 0)

{

TranslateMessage(&Msg);

DispatchMessage(&Msg);

}

* The message loop calls GetMessage(), which looks in your message queue. If the message queue is empty your program basically stops and waits for one (it *Blocks*).
* When an event occures causing a message to be added to the queue (for example the system registers a mouse click) GetMessages() returns **a positive value** indicating there is a message to be processed, and that it has filled in the members of the MSG structure we passed it. It returns 0 if it hits WM\_QUIT, and **a negative value** if an error occured.
* We take the message (in the Msg variable) and pass it to TranslateMessage(), this does a bit of additional processing, translating virtual key messages into character messages. This step is actually optional, but certain things won't work if it's not there.
* Once that's done we pass the message to DispatchMessage(). What DispatchMessage() does is take the message, checks which window it is for and then looks up the Window Procedure for the window. It then calls that procedure, sending as parameters the handle of the window, the message, and wParam and lParam.
* In your window procedure you check the message and it's parameters, and do whatever you want with them! If you aren't handling the specific message, you almost always call DefWindowProc() which will perform the default actions for you (which often means it does nothing).
* Once you have finished processing the message, your windows procedure returns, DispatchMessage() returns, and we go back to the beginning of the loop.

This is a very important concept for windows programs. Your window procedure is not magically called by the system, in effect *you call it yourself* indirectly by calling DispatchMessage(). If you wanted, you could use GetWindowLong() on the window handle that the message is destined for to look up the window's procedure and call it directly!

while(GetMessage(&Msg, NULL, 0, 0) > 0)

{

WNDPROC fWndProc = (WNDPROC)GetWindowLong

(Msg.hwnd, GWL\_WNDPROC);

fWndProc(Msg.hwnd, Msg.message, Msg.wParam, Msg.lParam);

}

However there are various issues such as Unicode/ANSI translation, calling timer callbacks and so forth that this method will not account for, and very likely will break all but trivial applications. So do it to try it, but don't do it in real code.

Notice that we use GetWindowLong() to retreive the window procedure associated with the window. Why don't we just call our WndProc() directly? Well our message loop is responsible for ALL of the windows in our program, this includes things like buttons and list boxes that have their own window procedures, so we need to make sure that we call the right procedure for the window. Since more than one window can use the same window procedure, the first parameter (the handle to the window) is used to tell the window procedure which window the message is intended for.

As you can see, your application spends the majority of it's time spinning round and round in this message loop, where you joyfully send out messages to the happy windows that will process them. But what do you do when you want your program to exit? Since we're using a while() loop, if GetMessage() were to return FALSE (that means 0), the loop would end and we would reach the end of our WinMain() thus exiting the program. This is exactly what PostQuitMessage() accomplishes. It places a WM\_QUIT message into the queue, and instead of returning a positive value, GetMessage() fills in the Msg structure and returns 0. At this point, the wParam member of Msg contains the value that you passed to PostQuitMessage() and you can either ignore it, or return it from WinMain() which will then be used as the exit code when the process terminates.

**IMPORTANT:** GetMessage() will return **-1** if it encounters an error. Make sure you remember this, or it will catch you out at some point... even though GetMessage() is defined as returning a BOOL, it can return values other than TRUE or FALSE, since BOOL is defined as UINT (unsigned int). The following are examples of code that may *seem* to work, but will not process certian conditions correctly:

**BRIEF HISTORY OF THE ORGANIZATION**

**Euro Information Technologies**

Euro Information Technologies provide solutions for a dynamic environment where business and technology strategies converge. Our approach focuses on new ways of business combining IT innovation and adoption while also leveraging an organization's current IT assets. We work with new generation technology companies - to build new products or services and to implement prudent business and technology strategies in today's dynamic digital environment.

It is a well-known fact that IT has a large role in bringing down the transactional costs in any organization. Euro Information Technologies looks at it from both, internal as well as external perspective for organizations. Euro Information Technologies believes that one aspect is incomplete without another.

Euro Information Technologies provides a complete range of solutions and services including IT infrastructure, network integration, training and system support.

The project was done for ‘Euro Soft’, a software development unit of ‘Euro Info Tech’. They launched several software products such as WinManager, Telecom Soft, and Human Resource Management Systems etc.

**DEFINITION OF PROBLEM**

**SYSTEM STUDY**

Once upon a time, software development consisted of a programmer writing code to solve a problem or automate a procedure. Nowadays, systems are so big and complex that teams of architects, analysts, programmers, testers and users must work together to create the millions of lines of custom-written code that drive our enterprises. The complexity is longer manageable by an individual alone, so teamwork is the order of the day.

To manage this complexity of the modern day software development, a number of system development life cycle (SDLC) models have been created: waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize.

The oldest of these, and the best known, is the **waterfall**: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following:

* **Project Planning and Feasibility Study:** This is the first stage of a basic software development life cycle. Establishes a high-level view of the intended project and determines its goals. At this stage the most important decision of “whether the project is feasible or not” is taken, for this the various aspects related with the project is taken into account and carefully analysed.
* **Systems Analysis and Requirements Definition:** System analysis is done by the System Analyst who is responsible the collection and processing of all the required data. The data is refined in order to convert project goals into defined functions and operation of the intended application. Analyses end-user information needs.
* **Systems Design:** Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudo code and other documentation. This stage of the software development is helpful in the determination of the basic structure and designs of the software so that the further planning, scheduling activities can be performed and the job division can be done.
* **Implementation:** This is the stage where the real code is written or the conversion of the software design into the actual programming language. The development is simplified by divided into the entire project into a number of modules so that cohesion is minimized and coupling is maximised. It also helps in the division of labour and scheduling.
* **Integration and Testing:** No project is completed unless a through and exhaustive testing is not completed. Testing is first done for all the modules separately and all the bugs and inconsistencies that correspond to those modules are trapped. The next step is the integration of all the modules together into a special testing environment, so that the errors, bugs, inconsistencies and interoperability problems that arise after the integration are trapped.
* **Acceptance, Installation, and Deployment:** First the Project has to be accepted by the client after an acceptance testing. Installation is the final stage of initial development, where the software is put into production and runs actual business. Here the end user gets the first actual feel of the system, and is in a situation to suggest improvements and inconsistencies.
* **Maintenance:** What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more. The Maintenance may be needed required due to the change in the operations of the system, advances in the technology, increase in the number of users etc. This is the least glamorous and perhaps most important step of all, goes on seemingly forever.

**Study and Analyse the Business Need**

The project team, supplemented by enterprise architecture or other technical experts, if needed, should analyse all feasible technical, business process, and commercial alternatives to meeting the business need. These alternatives should then be analysed from a life cycle cost perspective. The results of these studies should show a range of feasible alternatives based on life cycle cost, technical capability, and scheduled availability. Typically, these studies should narrow the system technical approaches to only a few potential, desirable solutions that should proceed into the subsequent life cycle phases.

The project team should develop high-level (baseline) schedule, cost, and performance measures, which are summarized in the System Boundary Document. These high-level estimates are further refined in subsequent phases. The acquisition strategy should be included in the SBD. The project team should determine the strategies to be used during the remainder of the project concurrently with the development of the CBA and Feasibility Study. Will the work be accomplished with available staff or do contractors need to be hired? Discuss available and projected technologies, such as reuse or Commercial Off-the-Shelf and potential contract types. The results of these assessments should be summarized in the SBD and documented in the Risk Management Plan and CBA. Estimate, justify, submit requests for, and obtain resources to execute the project in the format of the Capital Asset Plan and Justification, Exhibit 300.

**Existing System**

The existing system relies mainly on human effort. Separate registers are maintained for individual computer systems in the LAN and every purpose and information is gathered by going through these registers. Checking of existing records, editing and other reporting are done manually after monitoring the systems manually. When it is time to prepare the information about individual system performance and status, administrator first go through all the registers and collect the required information and after that, all the required calculations are performed manually (e.g. net free memory , free disk space, user account etc.). The report is prepared one record at a time.

**Limitations of the existing system**

The existing system, which is not fully computerised and is tedious and error prone. Since human effort is the backbone of the existing system, it is vulnerable in different areas. The main limitation is that it is time consuming. Here all works are done manually. If there are a lot of staff members, it is very difficult to prepare all the reports within the allotted time. It is also error-prone because it is very difficult to crosscheck all the reports every time. There is absolutely no security in the existing system. Anyone who manages to enter the room where all the registers are kept can access and modify any confidential data. Since the storage is in registers it requires a lot of space. Too much of data inconsistency and data redundancy is there in the current system.

**The Proposed System**

The proposed system is fully computerized. It provides a set of very simple interfaces for data input and report preparation. The system implements very strict security checking and data manipulation is allowed on the basis of user privileges. The main objective of the system is to provide a very simple and efficient interface for all the time consuming and tedious work. It is designed to be very user-friendly and any information can be collected very quickly at any time with the help of a few mouse clicks.

**Advantages of the new system**

The main advantage of the new system is its speed. Any report can be prepared at any time very quickly. It is highly reliable because it requires only very limited user-intervention. The project is highly customisable. It is generic system and can be customized according to the needs of the user. One of the important specialities of this system is that Server Component can be installed in any type of system running any one of the Windows Operating System Series.

**SYSTEM ANALYSIS**

The objective of system analysis phase is to collect all the requirements for the development. In this phase, the problem is studies carefully by dividing it into subsystems. The design of the system specifies the details that state how a system will meet the requirements identified during system analysis. Different techniques are used to collect the required information.

**Fact Finding**

Fact-finding is the first step in the initial investigation. It includes:

1. Review of written documents
2. On-site observation
3. Interview
4. Questionnaires

**Review of written documents**

All documents on data carriers (forms, reports, records, manuals, etc.) is organized and evaluated. Regarding existing forms, the analyst needs to find out how they are filled out, how useful they are to the user, what changes are to be made, and how easy they are to read. The project team went through different registers such as leave register, attendance register etc. to get the structure of the present data storage scheme. This step helped in the primary design of the database.

The current system uses a set of templates (basic forms) on paper for both input and output. The study of these forms helped in the design of both input and output screens.

**On-site observations**

Another fact-finding method used by the system analyst is on-site or direct observation. The analyst's role is that of an information seeker. During the analysis phase, the project team went to different schools to get an overall idea of the current system and information processing. This helped in the formation of the clear picture of the requirements of the end-user.

**Interview**

It is one of the most important and useful methods of information gathering. The project team interviewed the concerned staff and asked a set of previously prepared questions. Some of them are listed below.

1. How comfortable are you with the current system
2. What features do you like most in the current system
3. If you were allowed, what changes would you make to the current system?
4. What all features would you like to be included in the new system?

**Questionnaires**

It is a very simple and efficient method for information gathering. A set of questions were listed in the form a questionnaire and distributed to the end-users to get a clear picture of their view. Some of the questions are given below.

1. Do you think computerization will make your job easier?
2. Do you think the current method is obsolete?

**Feasibility Analysis**

Feasibility analysis is the procedure for identifying the candidate system, evaluating and electing the most feasible system. This is done by investigating the existing system in the area under investigation or generally ideas about a new system. It is a test of a system proposal according to its workability, impact on the organization, ability to meet user needs, and effective use of resources. The objective of feasibility study is not to solve the problem but to acquire a sense of its scope. Feasibility analysis involves 8 steps:

1. Form a project team and appoint a project leader.
2. Prepare system flow charts
3. Enumerate potential candidate systems
4. Describe and identify characteristics of candidate systems.
5. Determine and evaluate performance and cost effectiveness of each candidate system.
6. Weigh system performance and cost data.
7. Select the best candidate system.
8. Prepare and report the final project directive to management.

The key considerations that are involved in the feasibility analysis are economic, technical, behavioural and operational.

**Economic Feasibility**

Economic analysis is the most frequently used method for evaluating the effectiveness of a candidate system. It is more commonly known as cost benefit analysis, the procedure to determine the benefits and saving that are expected from a candidate system and compare them with costs. If the benefits outweigh costs then a decision is made to design and implement the system. Otherwise make alterations in the proposed system. We have chosen VB.NET and VC++.NET as the CLIENT and SERVER of the system respectively. VB.Net is the latest technology, which might be most helpful for future up gradations. This combination of VB.Net and VC++.NET Server is the most economically competent system for our purpose. The other combinations like Perl & Delphi etc. would be much costlier than the proposed system.

**Technical Feasibility**

The assessments of technical feasibility centres on the existing system and to what extent it can support the proposed addition. This was based on an outline design of system requirements in turns of inputs, files, programs, procedures, and staff. It involves financial considerations to accommodate technical enhancements. The present system is a partially automated system and the changeover is to a computerized system. Though it is a technical up gradation, highly technically qualified staff is not required to operate the designed system. Thus the existing staff can easily handle the resultant technical upgradation brought about by the proposed system. Thus the system is technically feasible.

**Behavioural Feasibility**

People are inherently resistant to change, and computers have been known to facilitate change. An estimate should be made about the reaction of the user staff towards the development of a computerized system. Computer installations have something to do with turnover, transfers and changes in job status. The introduction of a candidate system requires special effort to educate, sell and train the staff for conducting the business. The current system makes the job of the staff much easier and hence it is found to be feasible.

**Operational Feasibility**

The system is designed in a very user-friendly manner. Any layman with basic computer knowledge can use the system, with a little bit of training. Thus, the existing staff itself can manage the system after a short period of training. Recruitment of technically qualified staff for the purpose is not required.

The candidate system was found to be technically, economically, behaviourally and operationally feasible. The system was developed user friendly, needless training and improves the working environment. Disregarding the initial expenses, the candidate system was assessed to be feasible in all ways.

SYSTEM REQUIREMENT SPECIFICATION

**FRONT END : Visual Basic .NET**

BACKEND : Visual C++.NET

Hardware Requirements for WinMonitor Server

PROCESSOR : INTEL Celeron or INTEL

Pentium Series , AMD Processors.

CLOCK SPEED : 500 MHZ or Higher.

SYSTEM BUS : 16 BIT or Higher.

RAM : 32 MB or more.

Hard Disk Space : 500 MB or more.

Software Requirements for WinMonitor Server

OPERATING SYSTEM : WINDOWS 95/98/NT-Series.

LIBRARIES : WINDOWS common runtime routines,

MSXML.DLL etc.

Hardware Requirements for WinMonitor Client

PROCESSOR : PENTIUM III or Higher, AMD Athlon

or Higher

CLOCK SPEED : 500 MHZ or Higher.

SYSTEM BUS : 32 BIT or Higher.

RAM : 128MB

(196 MB Recommended)

Hard Disk Space : 10GB or more.

Software Requirements for WinMonitor Client

OPERATING SYSTEM : WINDOWS 98/XP/NT-Series (WINDOWS 2000SERVER

Recommended).

ENVIRONMENT : MICROSOFT .NET

MICROSOFT .NET FRAME WORK

**PROCESS LOGIC**

Since this is a system project, using flow diagrams like DFD (data flow diagrams) is inappropriate, instead we use UML(Unified Modelling Language). There are so many of them as given below.

1. State Diagrams.
2. Sequence Diagrams.
3. Class Diagrams.
4. Collaboration Diagrams.
5. Deployment Diagrams. e.t.c.

I used the second one, that is Sequence Diagrams, since they are very expressive in nature, if consider the project logic. It actually contains main Objects at the top represented with in rectangles. If the object is a person we user a human picture to represent that person. Then there are lines which represent the time dimension. Each thick rod on this line shows a full life line of the respected object at that time period. No such thick line shows the inactiveness of that object at that time period. Loops are represented by dotted rectangles with true conditions at the lower left of that rectangle with enclosing square brackets. The arrow shows flow of control or data from one object to another object.

If we use , it means control will transferred synchronously. That means the control will return to the object from which the arrow originated, only after the completion of the target object’s (To which the arrow pointing) current task. If we use , the control will return asynchronously. That means control will immediately return to the Arrow originating object, after the initiation of the target object.

Sequence diagrams will tell you the activities which will occurring during run time, so that we get a real picture of the working system.

**Remote**

**Admin**

[Loop: When ever Timer triggers]

If n/w available send to suitable remote mailbox and/or to, a remote computer System

if it is available

Objects Saved

Save Objects to File

Received Objects to Remote Admin

Objects Requested

Request Desktop View

Request Process List

An Event Occurred

**Local User**

**Network Monitor**

**Time**

**Monitor**

**Win**

**Monitor Server**

**Local**

**System**

Login

Load

Load needed DLLs

Loaded

Set Message Hooks

Set n/w Interface to Listen State

Set Timers

[Loop: When ever an event occurs]

Timer Expired

Reset Timer

Save Event to File

Event Saved

**Local User**

**Network Monitor**

**Time**

**Monitor**

**Win**

**Monitor Sever**

**Local**

**System**

Remote Admin Incoming Connection Info

Set up Buffers

Buffers allocated

Continue accepting incoming request

Properly Translated Objects

Translated request for system Info

Fetch Info

Requested Objects

[Loop: While Admin requests for System Info]

**Remote**

**Admin**

Objects to remote Admin

Connection request

Connection Termination Notification

Disconnect Request

Release Buffers

Request for local system Info

Authenticate

Disconnection Accepted

Disconnection Ok

[Loop: When ever a connection request encounters]

Logout

<<Destroy>>

<<Destroy>>

<<Destroy>>

**Network Monitor**

**Win**

**Monitor Client**

**Admin**

**Comp.System**

Incoming WinMonitorServer’s Info

Set up Buffers

Login

Initiate

Setting n/w interface to listen mode

Properly processed, received Information

Information Save request

Information Saved

Display Remote system info -rmation to user

Continue accepting incoming request

Buffers allocated

[Loop: While Remote WinMonitorServer requests for transferring SystemInfo

Disconnect Request

Transferring

Remote system Info

Connection request

Authenticate

**Remote Win Monitor Server**

**Win**

**Admin**

Connection Termination Notification

Release Buffers

Disconnection Accepted

Disconnection Ok

[Loop: When ever a connection request encounters]

**Network Monitor**

**Win**

**Monitor Client**

**Admin**

**Comp.System**

Target Win Monitor’s Address+Relevant Info

Set up Buffers

Requesting to connect to a remote WinMonitor Server

Buffers allocated

Notify user, about connection

Results Save request

Results Saved

Queries about the Remote System

Properly translated requests

Properly translated results

Display results of Queries to user

Results of Quer-ies

[Loop: While User having Queries to be made]

Send to Remote Host

Connection Status (Succeeded)

Connection request

Connection Ok

**Remote Win Monitor Server**

**Win**

**Admin**

Disconnect Request from user

Disconnect request

Disconnect Ok

Disconnection Accepted

Release Buffers

Connection Termination Notification

[Loop: When ever user like to connect to a remote WinMonitor server]

**Network Monitor**

**Win**

**Monitor Client**

**Admin**

**Comp.System**

Save Created EXE in disk

Requesting to create a WinMonitor Server for a particular Computer System

Intimation to user

EXE saved

**Remote Win Monitor Server**

Log Out

<<Destroy>>

[Loop: When ever user want to create a WinMonitor server for a particular host]

**Win**

**Admin**

<<Destroy>>

**SYSTEM DESIGN**

System Design is a process by which we develop the PROPOSED-SYSTEM through a sequence of interdependent sub processes. Therefore System Design is categorized in to four sub designing processes namely:

1. *DESIGN OF MODULES.*
2. *DESIGN OF INPUT.*
3. *DESIGN OF OUTPUT.*
4. *DESIGN OF STORAGE/FILE.*

Actually design is a complex procedure, where the Architecture of Design or Architectural Design is the most important. I mainly used the CLIENT/SERVER architecture, with CLIENT and SERVER were developed using MULTI-TIER architecture as depicted in the following diagrams.

**Architectural Design-WinMonitor Server**

**Server System OS**

**(Windows/OS family)**

**Server Core Layer**

**Client Interpreter**

**Server Network Interface**

Connection to remote client

**Architectural Design-WinMonitor Client**

**Client Presentation Layer**

**Client Application Layer**

**Server Interpreter**

**Client Network Interface**

Connection to remote server

**Client System OS**

**(WindowsNT/OS family)**

The entire server is coded in VC++.NET. The client part is implemented as a hybrid one, that means *Client Network Interface, Server Interpreter, Client Application Layer* are done in VC++.NET. *Client Presentation Layer* is fully implemented in VisualBasic.NET. One important thing is to note that the WinMonitorClient will need an OS from Windows-NT family, since the WinMonitorClient will perform editing within certain sections of WinMonitorServer exe template to make it adaptable for a particular host system. WinMonitorServer will work on ay Windows OS. Here by Network Interface, I mean that the interface used by the software to communicate through the network, which is heavily depend on the OS.

**MODULARIZATION**

It is the process of breaking the project into different smaller units. Modularisation helps in debugging of the modules involved in the project and also helps in reuse of the code. This helps in faster development, implementation and maintenance of the software. The project uses Object Oriented Terminologies for modularisation.

This type of program developing outwitted the older Monolithic approach to program development, where entire program is implemented as a single module. As the today’s systems grew more and more complex this approach has failed in many ways. First of all this type of systems cannot be easily modified and such a modification needs tremendous effort since we have to scan the entire source code. So that’s about monolithic software.

First, I like to depict the modularisation I used, using diagrams given below.

**WinMonitor Server**

**(Local PC Unit)**

**WinMonitor Client**

**(Admin Unit)**

**WinMonitor**

**Outgoing requests Monitor**

**Incoming requests Monitor**

**Network Monitor**

**WinMonitor Server Controller**

**Screen Monitor Interpreter**

**Key**

**Monitor**

**Interpreter**

**General**

**System Interpreter**

**WinMonitor**

**Server Builder**

**WinMonitor Server Interpreter**

**WinMonitor Client**

**Process Monitor Interpreter**

**Local**

**System Interpreter**

**User**

**Queries Handler**

**Remote**

**Host Controller**

**Mouse Controller**

**General**

**System**

**Controller**

**Keyboard Controller**

**General**

**Controls Interpreter**

**Mouse Controls Interpreter**

**Keyboard Controls Interpreter**

**Host Controls Interpreter**

**User**

**Queries Interpreter**

**Process Monitor**

**Local**

**System Monitor**

**WinMonitor Client Interpreter**

**WinMonitor**

**Timer**

**Controller**

**General**

**System Monitor**

**WinScreen**

**Monitor**

**WinKeyboard**

**Monitor**

**WinMonitor Server**

**Utilities**

**WinMonitor Server**

**Incoming requests Monitor**

**Outgoing requests Monitor**

**Network Monitor**

The system called WinMonitor.1.0 is actually consists of two independent operational units namely;

* **WinMonitor Client.**
* **WinMonitor Server.**

**WinMonitor Client**

This unit again divided into following sub modules and their description is given aside.

* **WinMonitor Server Builder**

This module will create a new WinMonitor Server unit specially intended for a target local PC, whose working is to be monitored.

* **Network Monitor**

This module will handle all the network traffic, incoming and outgoing connections to a number of WinMonitor Servers, including establishment, maintenance and graceful shutdown of these connections.

* **WinMonitor Server Interpreter**

This unit will handle all the data coming from the remote WinMonitor Server. These data may be in encoded or compressed format. It can be of user defined specific data. All these data are appropriately processed by this unit so that it can be understood by human beings using GUI. This section again contains:

* 1. Screen Monitor Interpreter.
  2. Key Monitor Interpreter.
  3. General System Interpreter.

Here Screen Monitor Interpreter will handle or process incoming screen images from the WinMonitor Server, which can be offline or Online. These images may be scaled or compressed , so this unit must perform decompression and appropriate transformation necessary before presenting it to the user.

Key Monitor Interpreter will handle all the character keys from the server. These keys can be originated at server, in the following ways. First it can be from Windows Hooks, a thread which continuously send a key character when the user presses that key for logging user typed keys purposes. Second it can be from the built in WinMonitor Chat Engine which allows both the server and client user to interchange plain text of messages. Third it can be from the server itself, like it’s name, free memory information etc.

General System Interpreter will handle generic information from the WinMonitor Server, like computer name, user name, memory usage etc. This unit is called the Local System Interpreter. The other sub unit in this unit is Process Monitor Interpreter, which handles the collected details of running processes from the server.

* **WinMonitor Server Controller**

This unit will send control information to remote WinMonitor Server to control that machine. It again subdivided into;

* 1. User Queries Handler.
  2. Remote Host Controller.

Here User Queries Handler will receive user instructions, such as execute a program and converts it to a proper format that WinMonitor Server understand, then send it to appropriate WinMonitor Server where the execution will be carried out.

Remote Host Controller will allow us to control the remote machine with our keyboard and mouse. We can see the desktop of the remote machine in our program and we can click on it or can press a key. These events will be taken and processed by this unit and will be send to the remote machine where these events will be carried out.

For these purposes Remote Host Controller contains three subsystems namely, KeyBoard Controller, Mouse Controller, General System Controller.

**WinMonitor Server**

This unit again divided into following sub modules and their description is given aside. Actually for every module in WinMonitor Client unit we find a peer part with in the WinMonitor Server unit.

* **WinMonitor Add-In Manager**

This module will acts like an add-in manager. That means with the help of this unit WinMonitor Client can upload a new WinMonitor Server Plug-in say Web Cam capture module and make that module functional so that it works like it has been an integral part of the WinMonitor Server, when the server is created.(not shown in the fig.).

* **Network Monitor**

This module will handle all the network traffic, incoming and outgoing connections to a number of WinMonitor Clients, including establishment, maintenance and graceful shutdown of these connections.

* **WinMonitor Server Utilities**

This unit will handle all the data outgoing from the WinMonitor Server. These data may be in encoded or compressed format during transfer. It can be of user defined specific data. All these data transformations are appropriately carried out by this unit so that it can be understood by WinMonitor Clients. This section again contains:

* 1. WinScreen/WinDesktop Monitor.
  2. WinKey Monitor.
  3. General System Monitor.

Here WinScreen Monitor will handle or process incoming request for screen images from the WinMonitor Clients, which can be offline or Online. These images may be scaled or compressed , so this unit must perform compression and appropriate transformation is necessary before transferring it to the WinMonitor Clients.

WinKey Monitor will log all the character keys typed by the local user. There is two cases, one during the initialization of the WinMonitor Server, it installs a Journal Hook to trap user typed keys for authentication purposes. This is used to determine which keys are typed by whom and where he types it. For ex. This technique is quite useful to know about the internet sites, the user visits by typing the site name at the explorer. The second case is when the administrator starts a chat section with the local user. These keys are also captured and send by this unit at the server side.

General System Monitor will gather and sends generic information from the Local computer where the server resides, like computer name, user name, memory usage etc. This unit is called the Local System Monitor. The other sub unit in this unit is Process Monitor, which will collect the details of running processes in the Local computer and send it to the WinMonitor Client.

* **WinMonitor Timer Controller**

This unit controls all the timer activities such as tries to reverse connect to the WinMonitor Client at regular intervals of time, Saving the current desktop image at predefined intervals of time etc.

* **WinMonitor Client Interpreter**

This unit will receive control information from remote WinMonitor Clients to control the Local machine, where the WinMonitor Server resides. It again subdivided into;

* 1. User Queries Interpreter.
  2. Remote Host Interpreter.

Here User Queries Interpreter will receive user instructions from remote WinMonitor Clients, such as execute a program and converts it to a proper format that the server operating system can understand, then execute it on the Local machine and send results back to the respected WinMonitor Client.

Remote Host Interpreter interprets operating system commands from remote WinMonitor Clients, such as mouse clicks, key presses etc, occurred at the Client side and try to execute that events in the Local machine where the WinMonitor Server resides. After executing the event it will then send the result, especially the change in the desktop will be forwarded to the respective WinMonitor Client with some transformations.

For these purposes Remote Host Controller contains three subsystems namely, KeyBoard Controls Interpreter, Mouse Controls Interpreter, General System Controls Interpreter. This is what called the Remote Desktop Controlling with respect to the WinMonitor Client side.

**INPUT DESIGN**

It is the process of converting user-originated inputs to a computer-based format. In the system design phase, the expanded data flow diagram identifies logical data flows, data store, sources, and destinations. Inaccurate input data are the most common cause of errors in data processing. Errors entered by data entry operators can be controlled by input design.

**Input Data**

The goal of designing input data is to make data entry as easy, logical, and free from errors as possible. In entering data, operators need to know the following:

1. The allocated space for each field.
2. Field sequence, which must match that in the source document.
3. The format in which data fields are entered.

Collection of input data is the most expensive part of the system, in terms of both equipments used and the number of people involved. While designing the form, great care is to be taken to simplify the work of the person filling it, minimize the errors made and also to reduce the effort of the person reading the form.

Data enters the system as input and this is the data on which the process is performed. It is necessary to ensure that the input design is suitable. A well -designed input should serve the following purposes:

1. To control work flow
2. To reduce redundancies in recording data
3. To increase clerical accuracy
4. To allow easier checking of data
5. The input is the link that ties the system with its users

In the system design phase the expanded data flow diagram identifies logical data flows, data sources, stores and destination. The objectives of input design must be effective, accuracy, and ease of use, consistency and simplicity.

Since my project is a system project, the number of inputs are considerably less in number, hence the efficiency is higher.

The main input section occurs during the building of a WinMonitor Server for a particular machine. There are number of configurations are to be set at that time. After building the user inputs are very less in number. It includes specifying an IP and Port Number to connect to a WinMonitor Server, entering password, specifying a exe name and it’s command line arguments for execution at the remote machine. All the above specified ones are the only text based inputs required after building a Server.

All other inputs include mouse clicks and command button clicks which reduced human input errors normally occurs during text based inputs. I here include the main text based inputs during the creation of a WinMonitor Server with supporting information. For the sake integrity I designed these input sections into tabs and grouped under a general form.

**Key Monitor Input Options**

This tab allow us to configure logging key strokes by the user in various ways. You can set the path and name of the log file with extension. You can specify the file upload size. Additional information can be included with key strokes. The path and filename are tested for validity whenever user enters every character, that ensures that user can enter only valid path and files names. Also textual input is kept to a minimum.

**Screen Monitor Input Options**

This input tab allows us to set the directory to store screen shots in the remote system. You can also select the root directory. The picture taking interval can be provided either in hours, minutes or seconds.

**WinMonitor Server Input Options**

This input tab configures the WinMonitor server being build generically. It allows you to specify its internal name and external name, Directory to save the server, Its identification with in the windows registry, It’s IP if we uses static IP etc. You can also specify the registry setting such as always run the server at start up etc. You can also specify install directory for the Server.

**Network Monitor Input Options**

You configure the network interface of the Server using this tab. It includes Listen Port, Reverse connection port, Emailing configurations if any, Reverse connection domain name or Client’s domain name, Reverse connection attempt interval etc. All these inputs are validated when each single character entered, so they are full proof.

After the creation of server, the textual input from user is very minimum. As mentioned earlier it includes entering IP-Address and Port number to connect to a Server and Passwords for authentications. All other inputs are designed to be mouse and button clicks, so that human input errors are minimum.

### Validation Checks

The process of checking whether entered data meets certain conditions or limitations is called validation. The input validations that are enforced in this system was formulated considering the following:

* Whether it is mandatory that data must be entered in this field?
* What characters are valid/invalid for this field?
* For numeric fields, is there a high/ low range limit?
* Is there a list of valid values for this field?
* Is this a conditional field?

The system performs the following validation checks:

* + **Data type validation:** Whether the data entered is a valid data type that is set for the field.
  + **Length of the data:** Checks whether the length of the data is within the set range of the data type.
  + **Foreign key validation:** If the key entered is a foreign key value, the system checks if such a value exists as a primary key of which this field is a foreign key.
  + **Data format validation:** Whether the data entered is in the given format or not.

### OUTPUT DESIGN

The Output design comprises of the design of forms through which various types of outputs can be obtained from the system . The success or failure of software is decided by the integrity and correctness output that is produced form the system. Since this is a system project, the term report has a little importance. Here importance is given to visual representation of information especially system status using text and graphics.

So the Outputs are categorized or divided into two.

* Graphical Analyzers.
* Textual Analyzers.

**Graphical Analyzers**

These are visual representation of remote system information using graphic utilities especially Icons, PIE charts etc. A good example is to show the memory usage as a pie chart with one section represent free memory and the second one represents the memory usage.

Second one is the **Remote Explorer** which shows folders and files in the remote system as Icons. These Icons can be folders, files or other objects. Also even progress bars can be group in to this category, since they shows the current progress of the task such as copying files or folders.

There are a lot of them which are shown along with the sample screens later in this document. One important one is the **BMP(Bitmap Files)** using to store desktop or window images of the remote system. These files can be compressed or normal. Normal files have the extension .BMP and compressed ones having extension **.CBMP**.

**Textual Analyzers**

These are character based outputs to the user. Win Monitor uses a lot of textual outputs, which are given below.

* **Remote Platform Information**

These are information regarding remote system operating system and primary memory.

* **Main System Information**

These includes generic system information such as remote system name, remote user name who logged in, total free main memory, total memory usage, total virtual memory and the list just goes on…

* **Remote Chat Information**

These are logged chat sections with Server and Clients for later viewing by administrators. It can also be a live chat session

* **User Messages**

These are textual information displayed in message boxes. This can be warnings, general information, acknowledgements etc.

* **Key Logger Information**

These are text files containing formatted text, which shows user typed keys for logging purposes. This can also be Online key logging.

* **System Disk Storage Information**

This information includes the name of the drives in the remote system with extended information such as type of each drive (It can be a network mapped drive, floppy drive, hard drive…), capacity and free space information, its cluster and sector information etc.

**FILE DESIGN**

First of all, I like to say that I am not using a database for storage purposes. At first this statement will give you a bad impression about the data storage used by WinMonitor. But there are a lot of reasons behind this decision.

Since this is a system project and the main aim of this project is to monitor and partially control another remote system. So the WinMonitor Server must work on almost all Windows platforms with a lesser requirement for resources. So I designed the project in that way, but some usage of specific Windows API’s(Application Programmer’s Interface) made the serve incompatible for systems running Windows95 or having lower versions of Windows.

But the Server will work on all other Windows Operating Systems, like Windows98/XP/2000/2000 Server etc. So you see even I dismiss the use of database, the server is incompatible for lower version of Windows. Suppose , I have to use a database the resource requirement for the Server increases and it may not run even in the Windows98, if I use SQL-SERVER 2000 having version 7.0 or later.

So you see ,I only omitted database for the sake of platform independency and efficiency under Windows Operating Systems. Also the Server must use less amount of processing power and main and secondary memory storage but must run faster consistently. So if we use databases data access using queries may lower the performance of the Server and the system needs database providers. This demands more memory and processing power and hence less efficient.

I overcame the storage requirement by using disk files and these files can be accessed and managed by low level IO (input/output) Windows API’s, which are faster than data access using queries.

1. KeyLog File.

This file is for storing user typed keys for tracking the activities of the user. This file can be of .dat, .txt, .file extensions as the administrator like. Following describes the fields of this type of file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | FIELDNAME | TYPE | SIZE | DESCRIPTION |
| 1. | Current Window | Text | NA | Active Window Name |
| 2. | Date | Date | NA | Key pressed Date |
| 3. | Time | Time | NA | Key pressed time |
| 4. | User name | Text | NA | User name |
| 5. | Keys Typed | Text | NA | User Typed Keys |

1. Process List File.

This file is used to keep track the processes spawn by the user. This file is not often stored in the disk, rather buffered in main memory and will be forwarded to the administrator. It‘s format is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | FIELDNAME | TYPE | SIZE | DESCRIPTION |
| 1. | Process Name | Text | NA | Name of the process |
| 2. | Process ID | Number | NA | PID of the process |
| 3. | Priority | Number | NA | Priority of process |
| 4. | Date | Date | NA | Process started Date |
| 5. | Time | Time | NA | Process started time |
| 6. | User name | Text | NA | User name |
| 7. | Memory Usage | Number | NA | Used Memory |
| 8. | Opened Files | Collection | NA | Opened files |

1. Screen Shot File.

This file contains screenshots which are taken at a regular intervals of time with the intension that to be forwarded to the administrator.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | FIELDNAME | TYPE | SIZE | DESCRIPTION |
| 1. | Bitmap data | Binary | NA | Bitmap file contents |

1. System Information.1 File.

This file contains information about the local system in general, like hardware and software configurations of the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | FIELDNAME | TYPE | SIZE | DESCRIPTION |
| 1. | System Name | Text | NA | Computer Name |
| 2. | OS Info | Text | NA | Name & Version of OS |
| 3. | HW Info | Text | NA | Hardware description |
| 4. | HDD Capacity | Number | NA | HDD storage capacity |
| 5. | Main Memory | Number | NA | Total Main memory |

1. System Information.2 File.

This file contains information about the local system which is changing, like available HDD memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SNO | FIELDNAME | TYPE | SIZE | DESCRIPTION |
| 1. | Local Drive | Text | NA | Local HDD drive name |
| 2. | Drive Size | Number | NA | Capacity of Drive |
| 3. | Free Space | Number | NA | Free Space available |

**SYSTEM TESTING**

Testing is a process of executing a program with the intention of finding an error. If testing is conducted successfully, it uncovers errors in software. Also testing demonstrates that software functions appear to be working according to the specification, that behavioral and performance requirements appear to have been met. Any software development organization or team has to perform several processes.

Software testing is one among them. It is the final opportunity of any programmer to detect and rectify any defects that may have appeared during the software development stage. Testing is a process of testing a program with the explicit intention of finding errors that make the program fail. In short system testing and quality assurance is a review in software products and related documentation for completion, correctness, reliability and maintainability.

System testing is the first stage of implementation, which is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct the goal will be successfully achieved. A series of testing are performed for the proposed system before the system is ready for user acceptance testing.

**TESTING OBJECTIVES**

* A good test case is one that has a high probability of finding an as yet undiscovered error.
* A successful test is one that uncovers an as yet undiscovered error.

**METHODS ADOPTED FOR TESTING**

The methods adopted for testing are

* Unit testing
* Boundary Value Analysis
* Integration testing
* Validation
* Output testing
* User acceptance testing

**Unit Testing**

This is the first level of testing. In this different modules are tested against the specification produces during the design of the modules. Unit testing is done during the coding phase and to test the internal logic of the modules. It refers to the modules. It refers to the verification of single program module in an isolated environment. Unit testing first focuses on the modules independently of one another to locate errors.

After coding each dialogue is tested and run individually. All necessary coding where removed and it was ensured that all the modules are worked, as the programmer would expect. Logical errors found where corrected.

So, by working all the modules independently and verifying the outputs of each module in the presence of staff we conducted that the programs was functioning as expected.

**Boundary Value Analysis**

Since a greater number of errors tend to occur at the boundaries than in the ‘CENTER’, Boundary value analysis is conducted. It leads to a selection of test cases that exercise boundary values.

During boundary value analysis, we developed test cases designed with values just above and just below the range if it was specified by the input condition. Also test case that exercised minimum and maximum numbers if the input condition specified a number of values were developed. Values just above and below the minimum and maximum were tested.

**Integration Testing**

Data can be lost across an interface: one module can be adverse effort on another; sub functions when combined may not produce the desired major functions. Integration testing is a systematic testing for constructing the program structure. Conducting the tests is to uncover errors associated within the interface. The objective is to take unit tested to modules and build a program structure. All the modules are combined and tested as a whole. Here correction is difficult because the vast expenses of the entire program complicate the isolation of causes. Thus in the integration testing step, all the errors uncovered are corrected for the next testing steps.

**Validation Testing**

This provides the final assurance that the software meets the all the functional, behavioural and performance requirements. The software is completely assembled as a package. Validation succeeds when the software functions in a manner in which user wishes. Validation refers to the process of using software in live environment in order to find errors. During the course of validation the system failure may occur and sometime the coding has to be hanged according to the requirement. Thus the feedback from the validation phase generally produces changes in the software.

Once the application was made of all logical and interface errors, inputting dummy data ensured that the software developed satisfied all the requirements of the user. This dummy data is known as test case.

**Output Testing**

After performing the validation testing, the next step is output testing of the proposed system since no system could be useful if it does not produce the required output in the specific format. Asking the users about the format the required by them, tests the output generated or considered into 2 ways. One is on screen and another is printed format.

The output format on the screen found to be correct as the format was designed in the system design phase according to the user needs. For the hard copy also, the output comes out as the specified requirement by the user. Hence output testing does not result in any correction in the system.

**User Acceptance Testing**

Acceptance test refers to the acceptance of data into the system for processing. The acceptance test contributes to the consistency and smooth working of the system. The system under consideration is tested for users at a time of developing and making changes whenever required. This is done with regard to the following points:

 Input screen design

 Output screen design

 Format of ad-hoc report and other outputs.

Taking the various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and correction are also noted for future use.

**Test Results**

The knowledge about the test results is necessary to make further modifications. The following are the test results.

 Program testing revealed that the program is free from syntax and logical errors. The output where found correct.

* String testing revealed that all the modules of the system works properly under peak loads and the system is capable of handling as many records.

Acceptance test refers to the acceptance of data into the system for processing. This acceptance test contributes to the consistency and smooth working of the system. Some of the important acceptance tests are given below:

**SYSTEM IMPLEMENTATION**

The implementation process is nothing but installing the system into user’s environment. The first step in the implementation process is getting approval from the organization where the system is to be implemented. The last minute minor changes that are required by the user can also be done in this phase. By satisfying the team of verification of the institute, the system is handed over to the organization. The implementation plan consists of training plans, Equipment installation and a plan for conversion from old system to the new one. Maintenance is also a part of the implementation phase.

The implementation is the final stage and it’s an important phase. It involves the individual programming, system testing, user training and the operational running of developed proposed system that constitute the application subsystems. One major task of preparing for implementation is education of users, which should really have been taken place much earlier in the project when they were being involved in the investigation and design work. During the implementation phase system actually takes physical shape. In order to develop a system implemented, planning is very essential. The implementation phase of the software development is concerned with translating design specifications in to source code The user tests the developed system and changes are made according to the their needs. Our system has been successfully implemented. Before implementation several tests have been conducted to ensure that no errors are encountered during the operation. The implementation phase ends with an evaluation of the system after placing it into operation for a period of time. Implementation is the third phase of the system process.

In order to achieve the objectives and the expected performance the system has been developed in a highly interactive and user-friendly manner.

**TRAINING**

As the end users are very much unfamiliar with the computerized system an intense training program is required to make them aware of the advantages of this system. At first, an awareness program was conducted so that they get an idea about the new trends and their advantages. Later they are given training to use the system in the most efficient manner.

**EQUIPMENT INSTALLATION**

As software and the hardware platform necessary for this system was not currently available, the entire equipment installation was necessary. Installation means installing the software into the hard disk of the target computer and creating necessary database and required tables within that database, in the target computer.

Normally the software installing and creation of required system files, especially configuration XML files and INI files are carried out as two independent processes. But our system integrates both these process into a single one. For that it uses WINDOWS-REGISTRY for storing the details of the server and client components.

During the first time of execution of the software after installing the system into the target computer, it checks the Windows-Registry to check whether the required configuration files are created and saved for the software i.e. Client and Server. If not, it will automatically extract the required configuration setting from its own EXE and creates the required configuration files. Later this configuration files will be used by the software. Then it stores the software configuration setting(ex: run at system start up) details to the Windows-Registry for later retrieval and creates necessary registry keys within the main registry key entry allotted to it.

**TECHNICAL PROBLEMS UNDER INSTALLATION PROCESS**

*PROBLEMS UNDER* **WinMonitorClient** *Installation*

Client component is entirely implemented in Visual Basic.NET and hence you require .NET frame work and common language run time to run the Client component.

*PROBLEMS UNDER* **WinMonitorServer** *Installation*

Server component is written in VC++.NET with out using managed extension and hence it will run with out the .NET frame work and common language run time. But it require Windows98 or later. It will not run in Windows95/ME since it uses some advanced SHELL API’s that will not shipped with Windows95/ME.

**APPENDIX**

This is the last section of this documentation, and I use this occasion to conclude the project development cycle. I also introduce sample screens here to show the outputs of the software. Since this is a system project the term Report will loose its significance and I will say that the output screens will serve the purpose of Reports and I always tried to design the output screens to resemble the standard Reports coming with Business Software Packages.

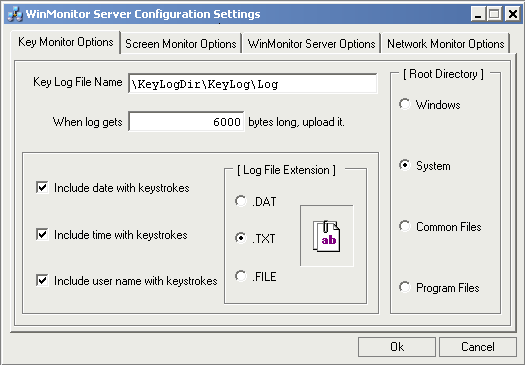
**CONCLUSION**

At this time of century this software is getting highly acclaimed. This system has been designed to meet the user requirements. All the suggestions are forwarded in the software proposal has been successfully completed and the final threshold of the application has been successfully completed. Viewing through the system developed a brief idea can be given as follows.

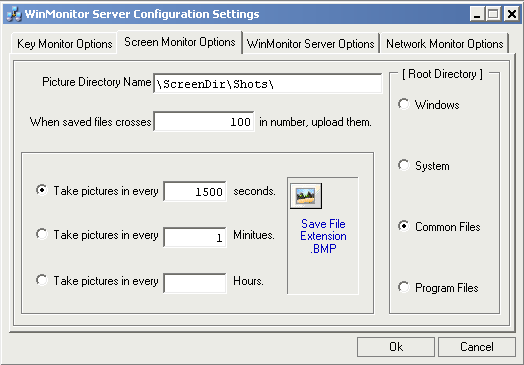
* Comprehending the problem
* Studying the existing system
* Building up the course of action to reach the goal
* Designing the problem
* Visualizing the solution as reports
* Preparing the screen outputs
* Testing the system with test data
* Achieving the required result
* Documenting the result

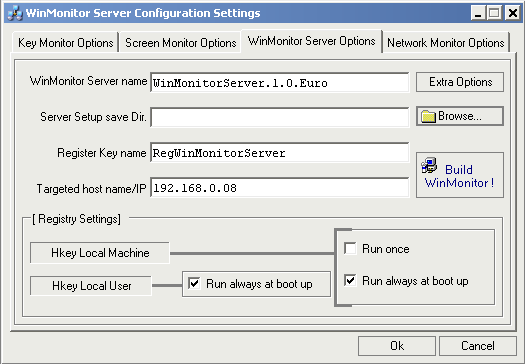
This system is more helpful and advantageous over the existing system. The entire system is menu assisted and highly interactive. In this system, neat formatted reports can be printed within a short period of time. The system is very user friendly and reports are screen oriented. Accurate updating, data validation and integrity are observed in the system. On trial run the performance and the efficiency calculated from the system was found to be almost satisfactory.

**SAMPLE SCREEN SHOTS**

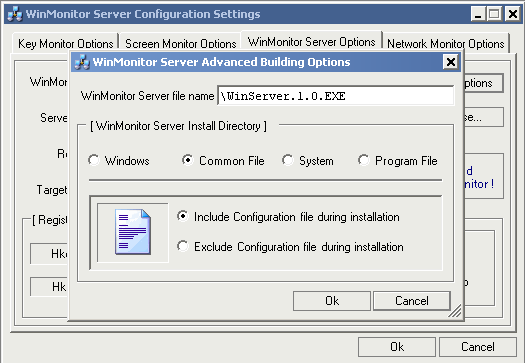
WinMonitorBuilder-Key Logger Settings ****

WinMonitorBuilder-Screen Monitor Settings

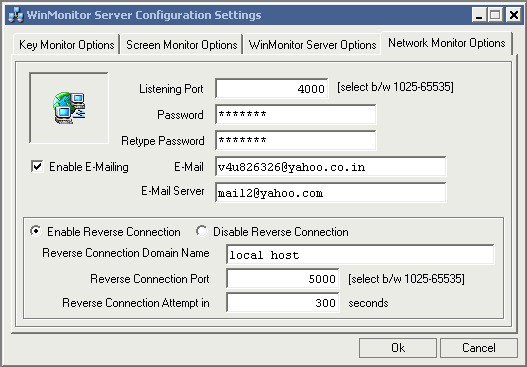
****WinMonitorBuilder-Win Monitor Server Settings

****

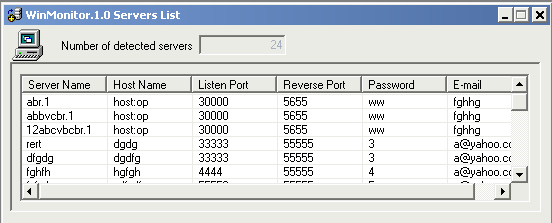
WinMonitorBuilder-Win Monitor Server Advanced Settings

****

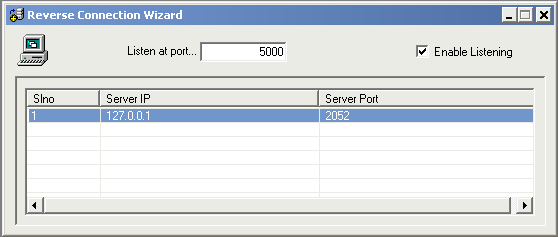
WinMonitorBuilder-Network Monitor Settings



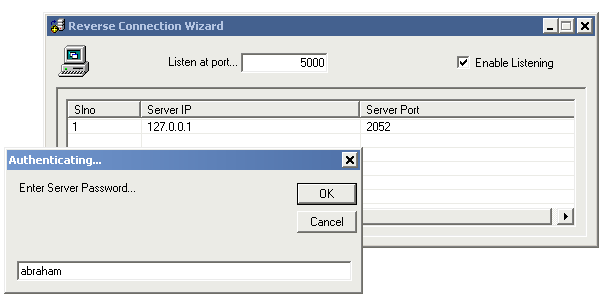
WinMonitor Client-Generated Servers Report



WinMonitor Client-Reverse Connection Wizard



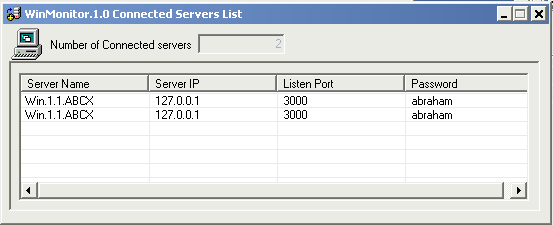
WinMonitor Client-Reverse Connection Authentication



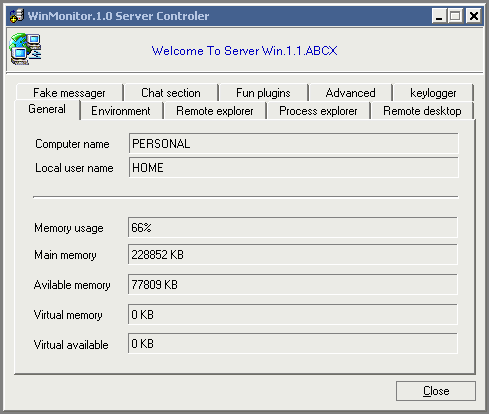
WinMonitor Client-Direct Connection Wizard



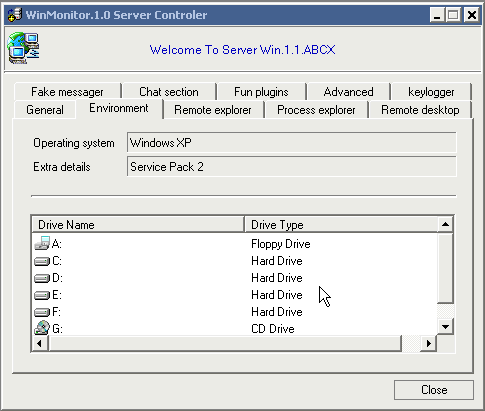
WinMonitor Client-Connected Servers List



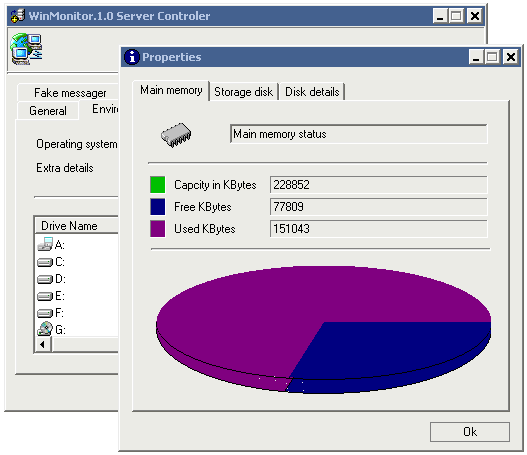
WinMonitor Client-General Remote System Information



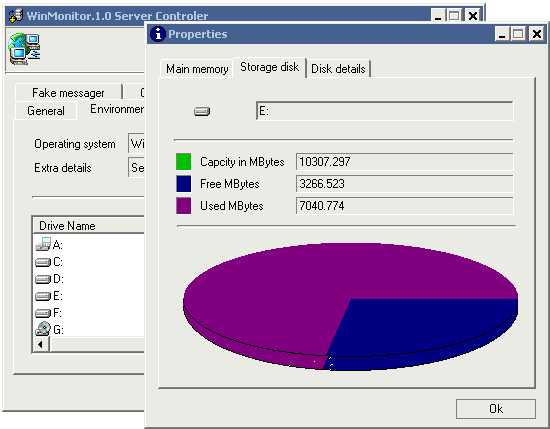
WinMonitor Client-Remote System Disk Information

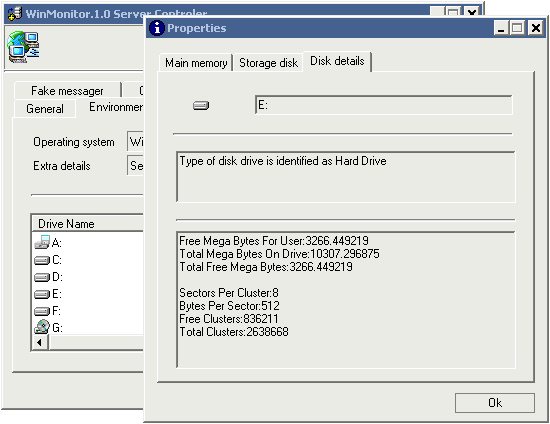


WinMonitor Client-Remote System Main Memory Advanced Information

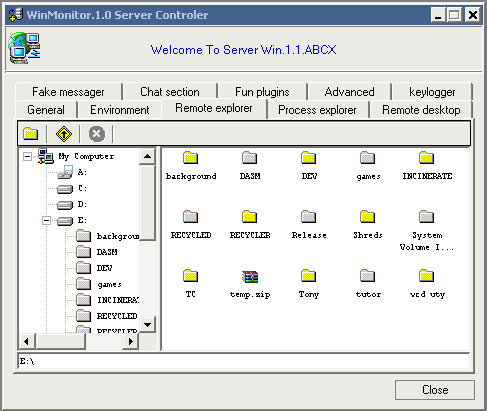


WinMonitor Client-Remote System Storage Drive Advanced Information

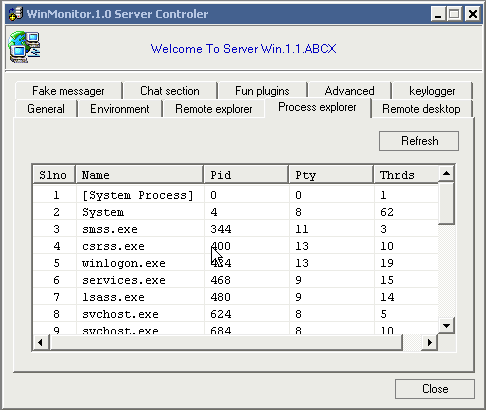




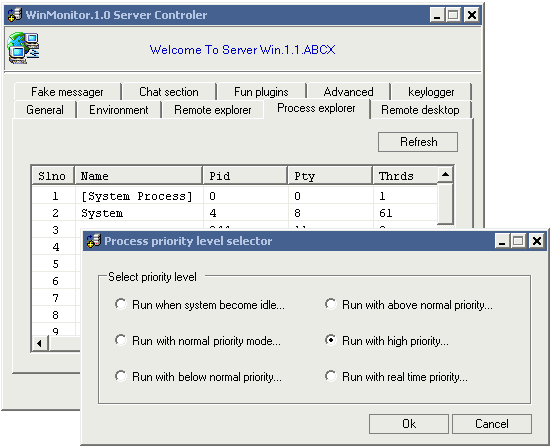
WinMonitor Client-Remote System Mini Explorer

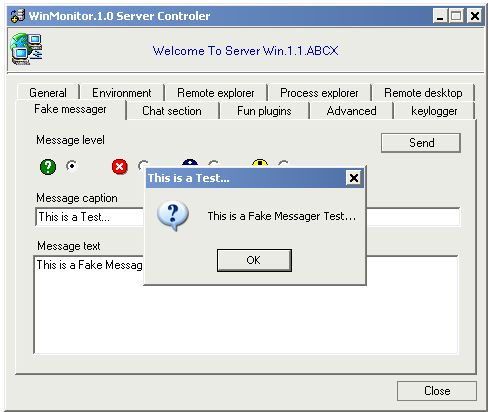


WinMonitor Client-Remote System Process Explorer

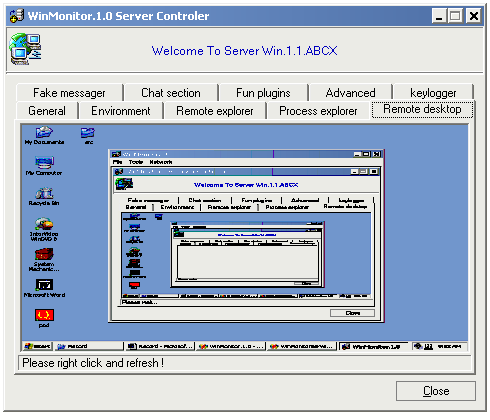
****

WinMonitor Client-Remote Process Priority Wizard and Fake Messenger

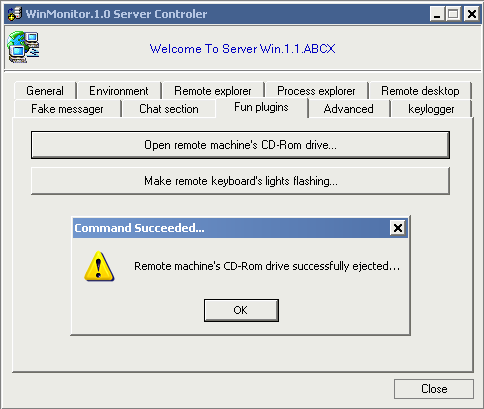
****

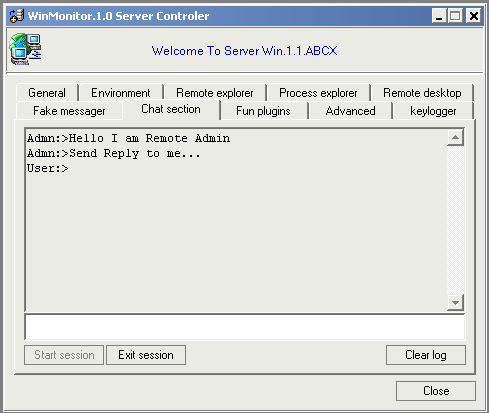


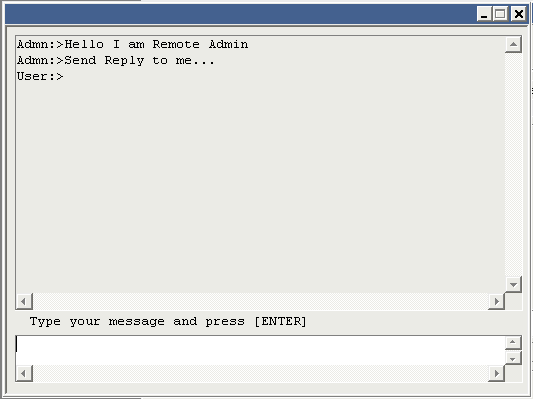
WinMonitor Client-Remote Desktop Explorer



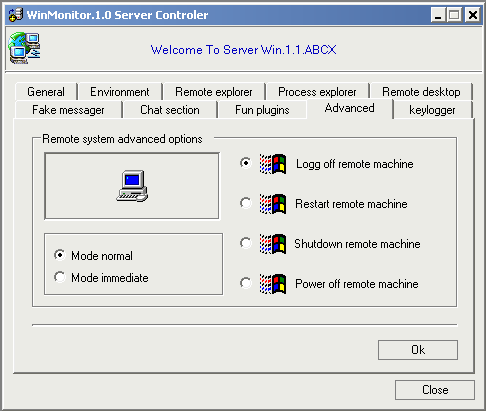
WinMonitor Client-Fun Plugins

WinMonitor Remote Chat-Server & Client Windows

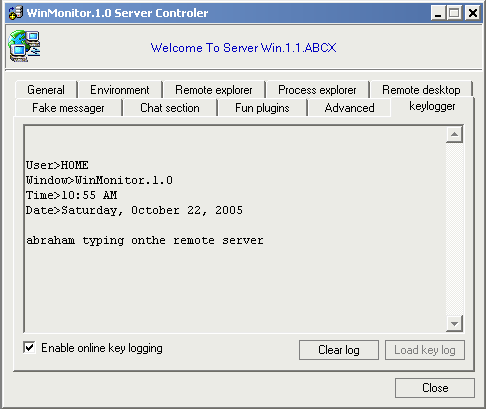




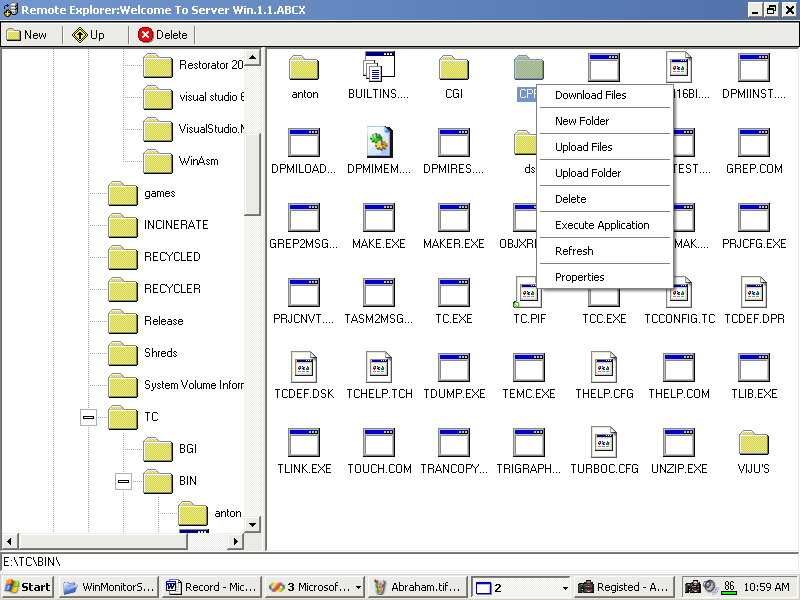
WinMonitor Client-Advanced Remote System Options

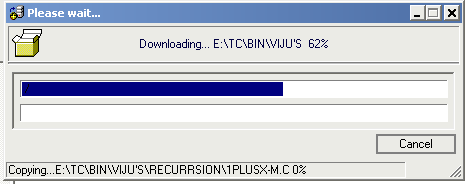


WinMonitor Client-Remote System On Line Key logging

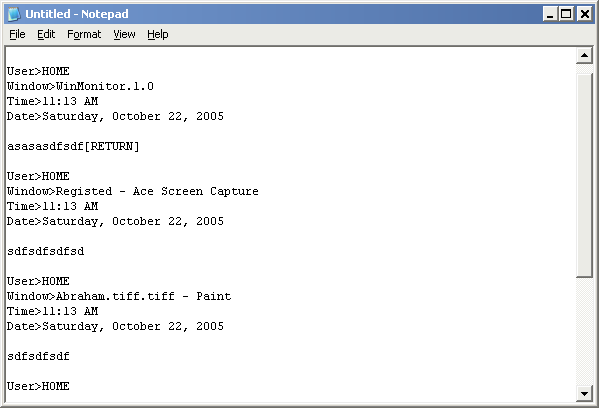


WinMonitor Client-Remote System Explorer Full View And Downloading

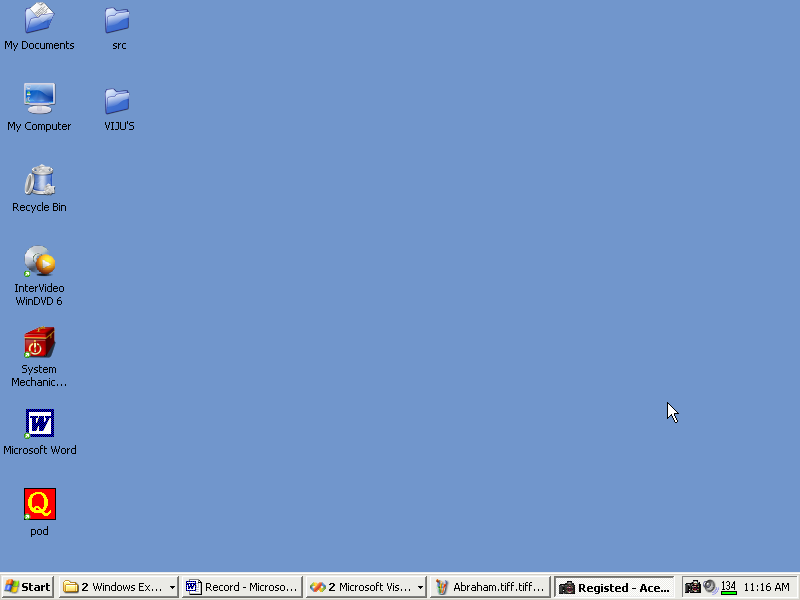




Offline Key Logging-Key Log File



WinMonitor Server-Offline Desktop Capture/Saved BMP File



**SAMPLE CODE**

**Visual C++.NET Sample Code**

#include<WINDOWS.H>

//-----------------WinMonitor Main Types------------------------//

#define WIN\_MONITOR\_NULL\_COMMAND 0x00

#define WIN\_MONITOR\_KEEP\_ALIVE 0x01

#define WIN\_MONITOR\_SHUTDOWN\_CONNECTION 0x02

#define WIN\_MONITOR\_GET\_PWD 0x03

#define WIN\_MONITOR\_GOT\_PWD 0x04

#define WIN\_MONITOR\_PWD\_OK 0x05

#define WIN\_MONITOR\_COMMAND\_MONITOR 0x06

#define WIN\_MONITOR\_SCREEN\_MONITOR 0x07

#define WIN\_MONITOR\_CHAT\_CLIENTSERVER 0x08

#define WIN\_MONITOR\_NORMAL\_CLIENTSERVER 0x09

#define WIN\_MONITOR\_CLIENT\_ACCEPTED 0x0A

#define WIN\_MONITOR\_GET\_CLIENTTYPE 0x0B

#define WIN\_MONITOR\_KEYLOG\_CLIENTSERVER 0x0C

//----------------Command Monitor Client/Server----------------//

#define COMMAND\_MONITOR\_NULL\_COMMAND 0x00000000

#define COMMAND\_MONITOR\_GET\_LOGICAL\_DRIVES 0x00000010

#define COMMAND\_MONITOR\_GET\_SUB\_DIRS 0x00000020

#define COMMAND\_MONITOR\_GET\_DRIVE\_TYPE 0x00000030

#define COMMAND\_MONITOR\_GET\_DRIVE\_INFO 0x00000040

#define COMMAND\_MONITOR\_GET\_FILE\_ASEXE 0x00000050

#define COMMAND\_MONITOR\_DOWNLOAD\_FILE 0x00000060

#define COMMAND\_MONITOR\_DISPLAY\_MSG 0x00000070

#define COMMAND\_MONITOR\_SET\_COMPRESSION 0x00000080

#define COMMAND\_MONITOR\_GOT\_LOGICAL\_DRIVES 0x00000090

#define COMMAND\_MONITOR\_GOT\_SUB\_DIRS 0x000000A0

#define COMMAND\_MONITOR\_GOT\_DRIVE\_TYPE 0x000000B0

#define COMMAND\_MONITOR\_GOT\_DRIVE\_INFO 0x000000C0

#define COMMAND\_MONITOR\_GOT\_FILE\_ASEXE 0x000000D0

#define COMMAND\_MONITOR\_DOWNLOADED\_FILE 0x000000E0

#define COMMAND\_MONITOR\_DISPLAYED\_MSG 0x000000F0

#define COMMAND\_MONITOR\_SET\_COMPRESSION\_OK 0x00000100

//----------------Additional Commands-----------------//

#define COMMAND\_MONITOR\_GET\_WINSERVER\_NAME 0x00000110

#define COMMAND\_MONITOR\_GOT\_WINSERVER\_NAME 0x00000120

#define COMMAND\_MONITOR\_GET\_MAIN\_MEMORY 0x00000130

#define COMMAND\_MONITOR\_GOT\_MAIN\_MEMORY 0x00000140

#define COMMAND\_MONITOR\_GET\_COMPUSER\_NAME 0x00000150

#define COMMAND\_MONITOR\_GOT\_COMPUSER\_NAME 0x00000160

#define COMMAND\_MONITOR\_GET\_OS\_VERSION 0x00000170

#define COMMAND\_MONITOR\_GOT\_OS\_VERSION 0x00000180

#define COMMAND\_MONITOR\_GET\_SUB\_DIR\_NAMES 0x00000190

#define COMMAND\_MONITOR\_GOT\_SUB\_DIR\_NAMES 0x000001A0

#define COMMAND\_MONITOR\_GET\_PROCESSLIST 0x000001B0

#define COMMAND\_MONITOR\_GOT\_PROCESSLIST 0x000001C0

#define COMMAND\_MONITOR\_KILL\_PROCESS 0x000001D0

#define COMMAND\_MONITOR\_KILLED\_PROCESS 0x000001E0

#define COMMAND\_MONITOR\_EXEC\_PROGRAM 0x000001F0

#define COMMAND\_MONITOR\_EXECD\_PROGRAM 0x00000200

#define COMMAND\_MONITOR\_SET\_PROC\_PRIORITY 0x00000210

#define COMMAND\_MONITOR\_SET\_PROC\_PRIORITY\_OK 0x00000220

#define COMMAND\_MONITOR\_LOGOFF 0x00000230

#define COMMAND\_MONITOR\_LOGOFF\_OK 0x00000240

#define COMMAND\_MONITOR\_SHUTDOWN 0x00000250

#define COMMAND\_MONITOR\_SHUTDOWN\_OK 0x00000260

#define COMMAND\_MONITOR\_POWER\_OFF 0x00000270

#define COMMAND\_MONITOR\_POWER\_OFF\_OK 0x00000280

#define COMMAND\_MONITOR\_RESTART 0x00000290

#define COMMAND\_MONITOR\_RESTART\_OK 0x000002A0

#define COMMAND\_MONITOR\_DELETE\_FILE 0x000002B0

#define COMMAND\_MONITOR\_DELETE\_FILE\_OK 0x000002C0

#define COMMAND\_MONITOR\_CREATE\_FOLDER 0x000002D0

#define COMMAND\_MONITOR\_CREATE\_FOLDER\_OK 0x000002E0

#define COMMAND\_MONITOR\_UPLOADED\_FILE 0x000002F0

#define COMMAND\_MONITOR\_FILE\_UPLOADED 0x00000300

#define COMMAND\_MONITOR\_LISTEN\_PORT 0x00000310

#define COMMAND\_MONITOR\_INSTALL\_KEYBOARD\_FUN\_PLUGIN 0x00000320

#define COMMAND\_MONITOR\_INSTALLED\_KEYBOARD\_FUN\_PLUGIN 0x00000330

#define COMMAND\_MONITOR\_UNINSTALL\_KEYBOARD\_FUN\_PLUGIN 0x00000340

#define COMMAND\_MONITOR\_UNINSTALLED\_KEYBOARD\_FUN\_PLUGIN 0x00000350

#define COMMAND\_MONITOR\_OPEN\_CDROM 0x00000360

#define COMMAND\_MONITOR\_OPENED\_CDROM 0x00000370

#define COMMAND\_MONITOR\_CLOSE\_CDROM 0x00000380

#define COMMAND\_MONITOR\_CLOSED\_CDROM 0x00000390

//---------------------------------------------------//

#define COMMAND\_MONITOR\_COMPRESS\_NIL 0x00000000

#define COMMAND\_MONITOR\_COMPRESS\_LZSS 0x00000001

#define COMMAND\_MONITOR\_GetType(uint32Header) ((uint32Header) & 0xFFFFFFF0)

#define COMMAND\_MONITOR\_GetCompression (uint32Header)((uint32Header) &

0x0000000F)

#define COMMAND\_MONITOR\_SetType(Choice,uint32Header) uint32Header=(((uint32Header) | 0xFFFFFFF0)& ((Choice) | 0x0000000F))

#define COMMAND\_MONITOR\_SetCompression(Choice,uint32Header) uint32Header=(((uint32Header) | 0x0000000F)& ((Choice) | 0xFFFFFFF0))

#define COMMAND\_MONITOR\_MSG\_INFO 0x01

#define COMMAND\_MONITOR\_MSG\_STOP 0x02

#define COMMAND\_MONITOR\_MSG\_QSTN 0x03

#define COMMAND\_MONITOR\_MSG\_EXCL 0x04

#define COMMAND\_MONITOR\_DRIVE\_UNKNOWN 0x00

#define COMMAND\_MONITOR\_DRIVE\_NOROOT 0x01

#define COMMAND\_MONITOR\_DRIVE\_REMOVABLE 0x02

#define COMMAND\_MONITOR\_DRIVE\_FIXED 0x03

#define COMMAND\_MONITOR\_DRIVE\_CDROM 0x04

#define COMMAND\_MONITOR\_DRIVE\_RAMDISK 0x05

#define COMMAND\_MONITOR\_DRIVE\_REMOTE 0x06

#define COMMAND\_MONITOR\_EXE\_UNKNOWN 0x00

#define COMMAND\_MONITOR\_EXE\_32BIT\_BINARY 0x01

#define COMMAND\_MONITOR\_EXE\_64BIT\_BINARY 0x02

#define COMMAND\_MONITOR\_EXE\_DOS\_BINARY 0x03

#define COMMAND\_MONITOR\_EXE\_OS216\_BINARY 0x04

#define COMMAND\_MONITOR\_EXE\_PIF\_BINARY 0x05

#define COMMAND\_MONITOR\_EXE\_POSIX\_BINARY 0x06

#define COMMAND\_MONITOR\_EXE\_WOW\_BINARY 0x07

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_NO\_CHANGE 0x00

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_ABOVE\_NORMAL 0x01

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_BELOW\_NORMAL 0x02

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_HIGH\_PRIORITY 0x03

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_IDLE\_PRIORITY 0x04

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_NORMAL\_PRIORITY 0x05

#define COMMAND\_MONITOR\_PROCESS\_PRIORITY\_REALTIME\_PRIORITY 0x06

#define COMMAND\_MONITOR\_SYSTEM\_DOWN\_NORMAL 0x00

#define COMMAND\_MONITOR\_SYSTEM\_DOWN\_IMMEDIATE 0x01

//-------------Screen Monitor Types & Comprn-------//

#define SCREEN\_MONITOR\_TYPE\_UNDEF 0x00

#define SCREEN\_MONITOR\_TYPE\_DIBITMAP 0x10

#define SCREEN\_MONITOR\_TYPE\_DIBITMAPFILE 0x20

#define SCREEN\_MONITOR\_COMPRESS\_NIL 0x00

#define SCREEN\_MONITOR\_COMPRESS\_LZSS 0x01

//-------------Screen Monitor Messages-------//

#define SCREEN\_MONITOR\_TYPE\_TAKE\_DESKTOP 0x30

#define SCREEN\_MONITOR\_TYPE\_TAKE\_WINDOW 0x40

#define SCREEN\_MONITOR\_TYPE\_SET\_COMPRESSION 0x50

//-------------------------------------------//

#define SCREEN\_MONITOR\_GETCOMPRESSION(Byte) ((Byte) & 0x0F)

#define SCREEN\_MONITOR\_GETTYPE(Byte) ((Byte) & 0xF0)

#define SCREEN\_MONITOR\_SETCOMPRESSION(Choice,Byte) (((Byte)| 0x0F) & ((Choice)|0xF0) )

#define SCREEN\_MONITOR\_SETTYPE(Choice,Byte) (((Byte)| 0xF0) & ((Choice) | 0x0F))

//----------------File Monitor----------------//

#define CFileMonitor\_FILENAME\_ONLY 1

#define CFileMonitor\_FILENAME\_AND\_SIZE 2

#define CFileMonitor\_FILEINFO\_DETAILED 3

#define CFileMonitor\_FILETYPE\_UNDEF 0

#define CFileMonitor\_FILETYPE\_DIR 1

#define CFileMonitor\_FILETYPE\_NORMAL 2

#define CFileMonitor\_FILETYPE\_HIDDEN 3

#define CFileMonitor\_FILETYPE\_TEMP 4

#define CFileMonitor\_FILETYPE\_ARCHIVE 5

#define CFileMonitor\_FILETYPE\_ENCRYPT 6

#define CFileMonitor\_FILETYPE\_SYSTEM 7

#define CFileMonitor\_FILETYPE\_READONLY 8

#define CFileMonitor\_FILETYPE\_OFFLINE 9

#define CFileMonitor\_FILETYPE\_COMPRESSED 10

#include "stdafx.h"

#include "resource.h"

#include "..\WinMonitorCLIB.1.0\NetworkDatatypes.h"

#include "..\WinMonitorCLIB.1.0\NetworkHeaderFormats.h"

#include "..\WinMonitorCLIB.1.0\GenericExtendedQueue.h"

#include "..\WinMonitorCLIB.1.0\GenericUtility.h"

#include "..\WinMonitorCLIB.1.0\GenericNetwork.h"

#include "..\WinMonitorCLIB.1.0\TcpNetworkMonitorDataSocket.h"

#include "..\WinMonitorCLIB.1.0\TcpNetworkMonitorListener.h"

#include "..\WinMonitorCLIB.1.0\LzssCompression.h"

#include "..\WinMonitorCLIB.1.0\GenericCommandMonitorUtility.h"

#include "..\WinMonitorCLIB.1.0\exeExtraEditor.h"

#include "..\WinMonitorCLIB.1.0\WinMonitorXmlUtility.h"

//--------------Plugin Headers-----------//

#include ".\PluginInterface\IPluginGeneric.h"

#include ".\PluginInterface\IFunPlugin.h"

#include ".\PluginInterface\IScreenMonitorPlugin.h"

#include ".\PluginInterface\IChatServer.h"

#include ".\PluginInterface\IOnlineKeyLogging.h"

//--------------End Plugin Headers-----------//

#include "WinMonitorConfigStructsAndGlobals.h"

#include "InitializeServerBasic.h"

#include "CommandMonitorServer.h"

#include "WinMonitorServicer.h"

#include "InitializeServer.h"

#include "WinMonitorServer.h"

int APIENTRY \_tWinMain(HINSTANCE hInst,HINSTANCE hPreInst,LPTSTR lpCmdLine,int nCmdShow)

{

if(\_tcscmp((TCHAR\*)lpCmdLine,\_T("")))

{

CGeneralMemoryUtility memuty;

DWORD prevpid;

PROCESSENTRY32 pe32;

TCHAR \*tchrBadFileName=0,\*tchrpid=0,\*tchrXml=0;

memuty.GetToken((TCHAR\*)lpCmdLine,"|",&tchrBadFileName,&tchrpid);

\_stscanf(tchrpid,"%d",&prevpid);

if(GetProcess(prevpid,pe32))

{

HANDLE HProcess=OpenProcess(PROCESS\_ALL\_ACCESS,true,prevpid);

try

{

if(TerminateProcess(HProcess,0));

{

CloseHandle(HProcess);

tchrXml=

AttachAnotherFileName(tchrBadFileName,DEF\_CONFIG\_XML);

DeleteFile(tchrBadFileName);

DeleteFile(tchrXml);

}

}

catch(...) {}

}

memuty.DeleteAll((void\*\*)&tchrBadFileName);

memuty.DeleteAll((void\*\*)&tchrpid);

memuty.DeleteAll((void\*\*)&tchrXml);

}

if(!IsXMLexists()) if(!ExtractXMLfromEXE()) return false;

if(!MapExeToProperPlace()) return false;

gbl\_hinstCurrentInstance=hInst,gbl\_intWndShowType=nCmdShow;

MethodRetBoolArgInt MShow=0;

if(GblStat\_HModulePlugins\_1\_0)

{

if(MShow=(MethodRetBoolArgInt)GetProcAddress(GblStat\_HModulePlugins\_1\_0,

"SetCmdShowType"),MShow)

MShow(gbl\_intWndShowType);

InstallKBrdHook=(MethodRetBoolArgVoid)

GetProcAddress(GblStat\_HModulePlugins\_1\_0,"InstallKeyBoardHook");

UnInstallKBrdHook=(MethodRetBoolArgVoid)

GetProcAddress(GblStat\_HModulePlugins\_1\_0,"UnInstallKeyBoardHook");

}

if(!LoadWinMonitorServerConfigurationFromXml

(gbl\_struct\_WinMonitorServerConfig)) return 0;

if(!CreateKeyLogFile()) return 0;

if(!CreateScreenShotDir()) return 0;

if(!UpdateRegistry()) return 0;

if(!SetKeyBoardHook());

if(!InstallScreenSnatcher());

CInitializeNetworkLibrary::LoadLibraryAndVersion();

if(!InstallReverseConnection());

CTcpNwMonitorListener lst;

lst.ListenAt(false,

(UINT)gbl\_struct\_WinMonitorServerConfig.SNetworkConnection.lngListenPort);

lst.SetListenState(true);

CTcpNwMonitorConnection conn;

CWinMonitorServicer \*ser;

MSG msg;

HACCEL hAccelTable=LoadAccelerators(hInst,"");

while(true)

{

if(PeekMessage(&msg,NULL,NULL,NULL,PM\_REMOVE))

{

if(msg.message==WM\_QUIT) break;

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

else if(lst.RemoveClient(conn))

{

if(ser=new CWinMonitorServicer(),ser->ServiceClient(conn,true,true))

gbl\_lnkdlst\_TcpClients.InsertAtBack((void\*)ser);

else delete ser;

}

else Sleep(100);

}

UnInstallKeyBoardHook();

UnInstallScreenSnatcher();

UnInstallReverseConnection();

if(GblStat\_HModulePlugins\_1\_0) FreeLibrary(GblStat\_HModulePlugins\_1\_0);

while(gbl\_lnkdlst\_TcpClients.DeleteFromFront((void\*\*)&ser))

{

if(ser->IsServicing()) ser->StopService();

delete ser;

}

CInitializeNetworkLibrary::CleanUpLibrary();

return 0;

}

**Visual Basic.NET Sample Code**

Imports WinMonitorBLIB\_1\_0

Public Class FrmStartUpConnection : Inherits System.Windows.Forms.Form

Private m\_struty As New CStringUtility()

Private m\_boolConnected As Boolean

Private m\_ConnectionAcceptThrd As System.Threading.Thread

Private m\_ConnectionAcceptThrdSub As System.Threading.Thread

Private m\_TcpConnection As CTcpNetworkMonitorConnection

Private m\_intStatus As Integer

Private m\_WinMessenger As CWinMonitorMessenger

Public Event eventGotConnection(ByRef TcpConnection As

CTcpNetworkMonitorConnection)

Private Sub FrmStartUpConnection\_Load(ByVal sender As System.Object,

ByVal e As System.EventArgs) Handles MyBase.Load

Me.txtConnectionServerPassword.Enabled = False

Me.btnServerPasswordOk.Enabled = False

Me.btnCancel.Enabled = False

Me.btnConnect.Enabled = True

Me.lblConnectionStatus.Text = "Status:ready..."

m\_ConnectionAcceptThrd = Nothing

m\_boolConnected = False

End Sub

Private Sub btnConnect\_Click(ByVal sender As System.Object, ByVal e As

System.EventArgs) Handles btnConnect.Click

If (Not m\_struty.ValidIP(txtConnectionServerIP.Text)) Then

MessageBox.Show(IDS\_IPADDRESS\_INVALID, Me.Text,

MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1)

txtConnectionServerIP.Focus()

SendKeys.Send("{END}")

Return

End If

Dim intPort As Integer = Val(txtConnectionServerPort.Text)

If (intPort < IDI\_LOWER\_PORT Or intPort > IDI\_UPPER\_PORT) Then

MessageBox.Show(IDS\_PORTNUMBER\_INVALID, Me.Text,

MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1)

txtConnectionServerPort.Focus()

SendKeys.Send("{END}")

Return

End If

Me.btnConnect.Enabled = False

Me.btnCancel.Enabled = True

m\_ConnectionAcceptThrd = New Threading.Thread(AddressOf

AcceptConnection)

m\_ConnectionAcceptThrd.Start()

End Sub

Private Sub AcceptConnection()

m\_boolConnected = False

m\_TcpConnection = New CTcpNetworkMonitorConnection()

m\_intStatus = IDI\_TRYING\_CONNECTION

m\_ConnectionAcceptThrdSub = New Threading.Thread(AddressOf

Respond)

m\_ConnectionAcceptThrdSub.Start()

While (Not

m\_TcpConnection.ConnectTo(txtConnectionServerIP.Text,

Val(txtConnectionServerPort.Text))) : End While

m\_intStatus = IDI\_MAINTAIN\_CONNECTION

m\_WinMessenger = New

CWinMonitorMessenger(m\_TcpConnection)

If Not m\_WinMessenger.SetConnection(m\_TcpConnection) Then

MessageBox.Show(IDS\_WIN\_SERVICER\_FAILED, Me.Text,

MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1)

Call btnCancel\_Click(Nothing, Nothing)

Call FrmStartUpConnection\_Load(Nothing, Nothing)

Return

End If

If Not m\_WinMessenger.SynchronizeServer() Then

MessageBox.Show(IDS\_SYNC\_SERVER\_FAILED, Me.Text,

MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1)

Call btnCancel\_Click(Nothing, Nothing)

Call FrmStartUpConnection\_Load(Nothing, Nothing)

Return

End If

Try

If Not (m\_ConnectionAcceptThrdSub Is Nothing) Then

If (m\_ConnectionAcceptThrdSub.IsAlive()) Then

m\_ConnectionAcceptThrdSub.Abort()

End If

End If

m\_ConnectionAcceptThrdSub = Nothing

Catch ex As Exception : End Try

Me.lblConnectionStatus.Text = "Status:connected ok..."

Me.btnConnect.Enabled = Me.btnCancel.Enabled = False

Me.btnServerPasswordOk.Enabled = True

Me.txtConnectionServerPassword.Enabled = True

Me.txtConnectionServerPassword.Focus()

Me.txtConnectionServerPassword.Focus()

End Sub

Private Sub Respond()

Dim intCount As Integer = 3

Me.lblConnectionStatus.Text = IIf(m\_intStatus =

IDI\_TRYING\_CONNECTION, "Status:connecting...", IIf(m\_intStatus =

IDI\_VALIDATING\_CONNECTION, "Status:validating user",

"Status:maintaining"))

While (True)

intCount += 1

If (intCount > 15) Then intCount = 0

Me.lblConnectionStatus.Text = IIf(m\_intStatus =

IDI\_TRYING\_CONNECTION, "Status:connecting...",

IIf(m\_intStatus = IDI\_VALIDATING\_CONNECTION,

"Status:validating user", "Status:maintaining"))

Dim intIndx As Integer

For intIndx = 0 To intCount Step 1

Me.lblConnectionStatus.Text &= IDS\_DOT

Next intIndx

System.Threading.Thread.Sleep(100)

End While

End Sub

Private Sub btnCancel\_Click(ByVal sender As System.Object, ByVal e

As System.EventArgs) Handles btnCancel.Click

Try

If Not (m\_ConnectionAcceptThrdSub Is Nothing) Then

If (m\_ConnectionAcceptThrdSub.IsAlive()) Then

m\_ConnectionAcceptThrdSub.Abort()

End If

End If

m\_ConnectionAcceptThrdSub = Nothing

Catch ex As Exception : End Try

Try

If Not (m\_ConnectionAcceptThrd Is Nothing) Then

If (m\_ConnectionAcceptThrd.IsAlive()) Then

m\_ConnectionAcceptThrd.Abort()

End If

End If

m\_ConnectionAcceptThrd = Nothing

Catch ex As Exception : End Try

Me.btnCancel.Enabled = False

Me.btnConnect.Enabled = True

Me.lblConnectionStatus.Text = "Status:aborted by user..."

End Sub

Protected Overrides Sub OnClosing(ByVal e As

System.ComponentModel.CancelEventArgs)

Try

Call btnCancel\_Click(Nothing, e)

Catch ex As Exception : End Try

End Sub

Private Sub btnServerPasswordOk\_Click(ByVal sender As

System.Object, ByVal e As System.EventArgs) Handles

btnServerPasswordOk.Click

m\_ConnectionAcceptThrd = New Threading.Thread(AddressOf

AuthenticateConnection)

m\_ConnectionAcceptThrd.Start()

btnServerPasswordOk.Enabled = False

m\_ConnectionAcceptThrd.Join()

RaiseEvent eventGotConnection(m\_TcpConnection)

Me.Close()

End Sub

Private Sub AuthenticateConnection()

m\_boolConnected = True

m\_intStatus = IDI\_VALIDATING\_CONNECTION

m\_ConnectionAcceptThrdSub = New Threading.Thread(AddressOf

Respond)

m\_ConnectionAcceptThrdSub.Start()

If Not m\_WinMessenger.Authenticate

(Me.txtConnectionServerPassword.Text) Then

MessageBox.Show(IDS\_CONNECTION\_AUTHENTICATION\_FAILED,

Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,

MessageBoxDefaultButton.Button1)

Me.lblConnectionStatus.Text = "Status:validation

failed..."

m\_TcpConnection = Nothing

Return

End If

Try

If Not (m\_ConnectionAcceptThrdSub Is Nothing) Then

If (m\_ConnectionAcceptThrdSub.IsAlive()) Then

m\_ConnectionAcceptThrdSub.Abort()

End If

End If

m\_ConnectionAcceptThrdSub = Nothing

Catch ex As Exception : End Try

Me.lblConnectionStatus.Text = "Status:validation success..."

Return

End Sub

Private Sub txtConnectionServerIP\_KeyDown(ByVal sender As Object,

ByVal e As System.Windows.Forms.KeyEventArgs) Handles

txtConnectionServerIP.KeyDown

If (e.KeyCode = Keys.Enter) Then SendKeys.Send("{TAB}")

End Sub

Private Sub txtConnectionServerPort\_KeyDown(ByVal sender As

Object, ByVal e As System.Windows.Forms.KeyEventArgs)

Handles txtConnectionServerPort.KeyDown

If (e.KeyCode = Keys.Enter) Then SendKeys.Send("{TAB}")

End Sub

Private Sub txtConnectionServerPassword\_KeyDown(ByVal sender As

Object, ByVal e As System.Windows.Forms.KeyEventArgs)

Handles txtConnectionServerPassword.KeyDown

If (e.KeyCode = Keys.Enter) Then SendKeys.Send("{TAB}")

End Sub

End Class

**BIBLIOGRAPHY**

* Visual Basic .NET Programming.

Bill Evjen and Jason Beres

Wiley Publications.

* Object Oriented Programming with Vb.NET.

J.P Hamilton

Galgotia Publications, 2nd Edition 2003.

* MSDN.

Microsoft Visual Studio.NET.

* Visual C++.NET Programming Bible.

Tom Archer & Andrew White Chapel.

* Waite Group’s Windows API Bible.

James L. Conger.

* “www.codeproject.com”.

Internet Website.