Productivity Assessment Tool

By

Edward P. Hamann

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

May 2002

Productivity Assessment Tool

by

Edward P. Hamann

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements
for
the Degree of Bachelor of Science
in Information Engineering Technology

Edward P. Hamann	Date
Shannon McClintock, Faculty Advisor	Date
Lawrence Gilligan, Department Head	Date

Acknowledgements

I would like to express my gratitude for the help I received from Bob Schlemmer and Shannon McClintock while writing this software. I would also like to express gratitude to Steven Roman, the author of Win32 API Programming, before I read his book I held little to no knowledge of API programming.

Table of Contents

List of Figures

Abstract

The Productivity Assessment Tool, or PAT, is a network application which allows employers to evaluate their employee’s computer usage based on the amount of time spent using specific applications. PAT records the current foreground application file name and the time it was being used. This data is then presented in a variety of report formats to help employers monitor the productive time of their employee’s computer usage.   PAT can run on any MS Windows operating system.

Productivity Assessment Tool

1. Statement of Problem

I have observed many people sitting at their desks day in and day out looking at their computer screens. Does this mean they are working? Does this mean they are spending their time being productive? Now that more workers have computers they have access to new ways of wasting or stealing time from their employers such as using unapproved software or surfing the Internet. I have noticed people in my workplace playing solitaire for up to an hour a day. Some may say, “What is wrong with that?” There is nothing wrong with the game Solitaire, but the time they spent playing the game is owned by the company, which pays them for their work. In essence the employee is *stealing* time from the workweek for which they are paid. I define theft of time as any time spent deviating from an employer’s goal during the hours that a company pays an employee.

Edward Hamann Sr. stated, in an interview, that his clients look for fast turnaround times on their jobs. He also mentioned that in the past his clerical employees would waste time while he was out of the office. After 27 years of owning his own business he knew how long it took to complete certain tasks, but he had no way of proving that his employees would become unproductive while he was out of the office, except for the occasional screen left on with Web pages up or Solitaire games. He didn’t want to accuse or reprimand anyone without sufficient evidence. However, if his assumptions were right then he was wasting payroll money on inefficient workers.(8)

In another interview with Randall Diekmeyer, Systems Administrator at PEDCo E & A Services Inc., I asked if there were people in his company who would waste time using programs which do not relate to the tasks which they are supposed to be accomplishing. His response was “yes, definitely.” I also asked him if he believed that it was a problem. To that he also answered “yes.”(3)

The two professionals I spoke with agreed that there was is problem with employees stealing company time that is not easily solved. There is software on the market that can record some information such as keystrokes and allow the user to look at the desktop of a remote user. Since tools for monitoring worker activity, like Surf Spy, and add in features of Netware 6, are already on the market I believe that is an indication that there is a legitimate need that can be solved through monitoring software.

1.1 Description and Intended Use

The Productivity Assessment Tool has been designed to allow a manager to determine what software their employees are using throughout the workday. Managers don’t have time to spend walking around the office looking over the shoulders of their employees to see if they are on task and doing this makes most people feel nervous or uncomfortable. The money that is used to pay a manager to monitor this way is defiantly a wasted resource. How does one solve this problem? The solution can come through software.

The Productivity Assessment Tool is designed to address this issue. It tracks the executable names of the foreground tasks being run on a Windows Operating System. It also tracks the start time of that foreground event. This information is sent to the manager’s computer where it is stored in an Access database. By using the server application the Manager can generate reports to display lists of what software their employees used throughout the day.

1.2 Solution of the Problem

There are software packages available that allow employers to monitor activities of their employees, but developed a software package that lets the administrator tailor the monitoring to his own business and analyze the data collected to create reports automatically which show the results in an easy to understand way automatically. This differs from current methods because the data is analyzed for the administrator based on criteria set by the administrator such as program names and client names.

Theft of time happens in every place of business, and is hard to track. I am offering an answer to the problem of wasted company time due to misuse of company computers. The software developed monitors the programs that are being run on each computer on a network, and then create reports based on that data for management to review.

1.3 User Profile

The intended users for this system consist of up to three categories, IT staff, management, and the person being monitored. Their involvement in the functions of this software is mostly non-technical.

1.3.1 IT Staff

This user is not necessary, but in some organizations it is the job of the IT staff to perform all software installations and even manage its use. The IT staff would have to install the client on all of the MS Windows workstations being monitored and install the server application on the workstation which reports will be generated from. Their level of IT literacy could be minimal, however their job description typically would make them more than qualified to perform their tasks relation to this software.

1.3.2 Management

Management needs basic windows program operation skill (i.e. ability to use a mouse and access to a printer). The person in this position uses the server application which prints out reports generated from the Access database.

1.3.3 Person Being Monitored

Their part in the operation of this software is completely non-technical. They do not do anything. The program is executed when the computer is turned on, and they never have to know that it is watching them. The program won't even show up in the windows taskbar.

1.4 Project Design

I used three of the four areas covered in the Information Engineering Technology program to create this product. The main focus was on programming, however, Database and Networking were also be needed to achieve my goal. Below is a listing of each area and how it is relevant to this project.

1.4.1 Programming

This software was written using Visual Basic 6.0. VB was chosen because of its ease of use when connecting to databases, creating user interfaces, controls and it was the main language taught in the IET program. The VB Winsock control was used for the networking component and several API calls were used to perform the client functions.

1.4.2 Database

An MS Access 2000 database is used to store the retrieved client information on the workstation that holds the server application. The users do not see this portion of the software. The database is accessed by the server application using an ODBC connection to store the retrieved client information and provide data for report generation. Access 2000 was chosen because of its ease of use and low cost.

1.4.3 Networking

Winsock was used to connect the server and client applications. This is a networking tool that runs over the TCP/IP protocol. The server polls each machine for its data and it is delivered using Winsock. Winsock was chosen because it is the only network medium of which I have a working knowledge of its uses with Visual Basic.

2. Component Design

2.1 Client Design

The client's primary function is to collect the executable name and start time of the application, which owns the foreground window. To do this I use a variety of API calls. At an interval of every twenty seconds it executes its Visual Basic code (See Appendix D) to check for a new foreground event. If a new event is found it gets a time stamp and continues checking every twenty seconds. Once a new foreground event is found the client writes to a text file by the last tasks executable name and start time by using sequential file access methods. This process will continue over and over again. The code needed to perform this function will have to be different for windows 9x machines vs. Windows NT. This is because the API call to get the file name is not available in the same libraries in the two OS architectures.

2.2 Database Design

The database is accessed by the server application. Its primary function is to store the data collected by the client, and provide the server application with data through queries. The design is simple, there are three tables, one to store the client information, one to store the program executable names and one to store all events recorded. The database complies with all three rules of Normalization. Each table has a unique identifier, the data is broken down into its smallest units and there isn't any redundant data in the database.

2.3 Server Design

The server is used to collect the data from the client apps, store it in the database, and access it through queries that create reports. The user interface is simple, there are only three windows. The main window (see figure 1) Controls the collection of events from the client machines.

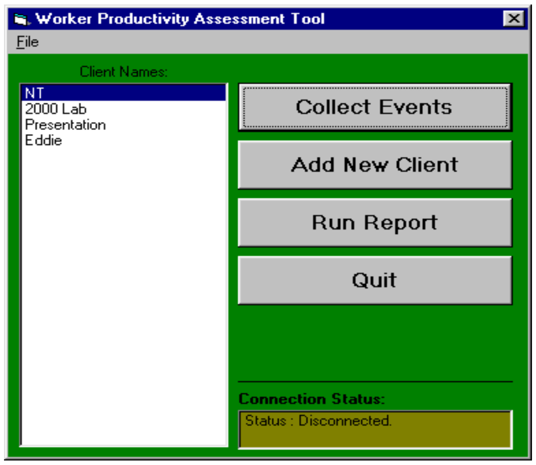


Figure 1. Main Window
(Screen shot of the main window)

On this screen there is a list box with all of the client names in it. To add new clients to the Client Names list box click on the Add New Client button. This button will open the Add New Client window (see figure 2) which enables the user to add all client data for the collection of events. The data required are a client name, and that client's IP Address. After entering the necessary information in the text boxes click add and the database will have a new entry for a client. After clicking close the Add Client Window will disappear and the new entries will be shown in the Client Names list box on the Main window.

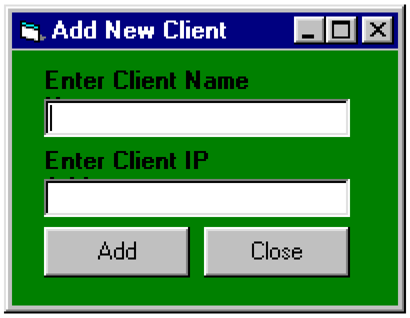


Figure 2. Add New Client Window
(Screen shot of Add New Client Window)

The user chooses from the computers in that list to acquire their foreground events history by clicking the Collect Events button. At the bottom right of the main window there is a label control which gives the user updates on the status of each collection of events (see figure 1. Page 6). Once all of the events have been loaded into the database the user is able to choose from the report format types by activating the Report Properties form by clicking the Run Report Button. When the button is pressed the Report Properties window opens (see figure 3 Page 8).

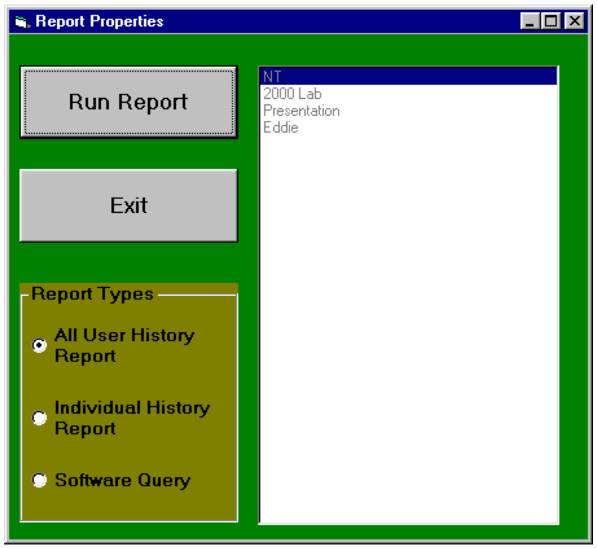


Figure 3. Report Properties Window
(Screen shot of the Report Properties Window)

For a list and description of each report see section 4.5 of this document.

2.4 Network Design

Winsock is used to send and receive information to and from the server. There are Windsock components in both the client and the server. When the client begins running it will wait and listen for the server. The server will initiate all communication. When commands are given from the server the client will execute code to send its data. Once the server has collected the data, the client will delete its logs and begin logging again until the next time the server collects data. Winsock requires TCP/IP to communicate across a network.

3. List of Deliverables

1. A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows 9x operating system.

2. A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows NT operating system.
3. A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows 2000 operating system.
4. A server application that collects data from clients using a Winsock connection and stores it in a MS Access database that conforms to the first three rules of normalization.
5. Three different report formats for the user to choose from which are created using the data environment object available using Visual Basic 6.0.
6. A timer set for twenty seconds for the collection or evaluation of new events.
7. Installation package for the client application.
8. Local storage of Client event data until server makes a Winsock connection and collects events.
9. Installation Package for the server application.

4. Proof of Design

4.1 Deliverable #1

4.1.1 Definition

A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows 9x operating system.

4.1.2 Description

The purpose of this deliverable was to get the executable name of the application running the active foreground window. This was accomplished through the use of several windows APIs (Application Programming Interface). APIs are functions that are used by the windows operating system which are open for use by other applications. The APIs are contained in files called DLLs (Dynamic Link Libraries). This deliverable required the use of seven API calls from two separate DLLs: Kernel32.dll, User32.dll. The API calls returned values of type long (numbers which range between –2,147,483,648 and 2,147,483,647), which I loaded into variables and used to call subsequent APIs. The final API call returned a string value which is the program name. For an example on how to do this with Visual Basic see **Appendix D.**

4.2 Deliverable #2

4.2.1 Definition

A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows NT operating system.

4.2.2 Description

The purpose of this deliverable was to get the executable name of the application running the active foreground window. This was accomplished through the use of several windows APIs (Application Programming Interface). APIs are functions that are used by the windows operating system which are open for use by other applications. The APIs are contained in files called DLLs

(Dynamic Link Libraries). This deliverable required the use of six API calls from three separate DLLs: Kernel32.dll, User32.dll and PSAPI.dll. The API calls returned values of type long (numbers which range between –2,147,483,648 and 2,147,483,647), which I loaded into variables and used to call subsequent APIs. The final API call returned a string value which is the program name. For an example on how to do this using Visual Basic see **Appendix E.**

4.3 Deliverable #3

4.3.1 Definition

A client application written in Visual Basic 6.0 that collects the executable name of the program that is currently the foreground window on a Windows 2000 operating system.

4.3.2 Description

This deliverable was solved in the exact same segment of code which made deliverable number two operate. This is because Windows 2000 was based off of the Windows NT 4.0 architecture and contains the same APIs which are located in DLLs with the same file names as Windows NT 4.0.

The purpose of this deliverable was to get the executable name of the application running active foreground window. This was accomplished through the use of several windows APIs (Application Programming Interface). APIs are functions which are used by the windows operating system that are open for use by other applications. The APIs are contained in files called DLLs (Dynamic Link Libraries). This task required the use of six API calls from three separate DLLs; Kernel32.dll, User32.dll and PSAPI.dll. The API calls returned values of type long, (numbers which range between –2,147,483,648 and 2,147,483,647) which I loaded into variables and used to call subsequent APIs. The final API call returned a string value which was the program name. For an example on how to do this using Visual Basic see **Appendix E.**

4.4 Deliverable #4

4.4.1 Definition

A server application that collects data from clients using a Winsock connection and stores it in a MS Access database that conforms to the first three rules of normalization.

4.4.2 Description

This deliverable has two separate attributes. First the collection of data from the clients and second the storage in an Access database.

The first attribute is the collection of data from the client applications. This is established using a Winsock connection. The Winsock connection is established using the Winsock control provided by the Visual Basic development environment. This control creates a connection between two computers. To establish this connection two pieces of information are needed: a port number and an IP Address. A port number describes a virtual location that is used to set up a connection between two computers and is represented by a number between 0 and 65,535. An IP Address is a unique identification number used on networks running TCP/IP. The IP Address is the only piece of information needed by the user of the server application. It is stored in the database in the Client Names Table (see figure 4. page 13). The software generates the port numbers when a connection attempt is executed.   Once the connection is established the client application transfers its data from a text file, described in deliverable 8, character by character until it creates a duplicate copy on the computer running the server application. Once the text file is copied the connection between the client and server is terminated.

Upon termination of the Winsock connection the text file is loaded into a database that conforms to the first three rules of normalization. The first three rules of normalization are; 1) each field in a table should represent a unique type of information., 2) Each table must have a unique identifier, or primary key, that is composed of one or more fields in the table., 3) For each unique primary key value, the values in the data columns must be relevant to, and must completely describe, the subject of the table. The text file is taken line by line and added to the ‘Event_Log’ table in the database which has a relationship to the ‘Client_Names’ table to indicate who the event belongs to using an ADODB connection (see figure 4).

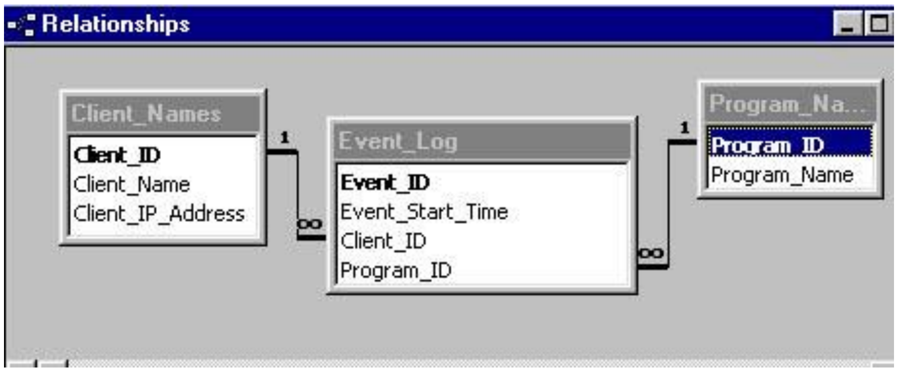


Figure 4. Database Tables
(Screen shot of Database Tables and Relationships)

4.5 Deliverable #5

4.5.1 Definition

Three different report formats for the user to choose from which are created using the data environment object available using Visual Basic 6.0.

4.5.2 Description

4.5.2.1 All User History Report

This report shows the complete history of software usage for every user with events reported to the database. It was created using the MS Visual Basic Report Designer. It is populated by the following SQL statement: SELECT Client_Names.Client_Name, Program_Names.Program_Name, Event_Log.Event_Start_Time FROM Program_Names INNER JOIN (Client_Names INNER JOIN Event_Log ON Client_Names.Client_ID = Event_Log.Client_ID) IN Program_Names.Program_ID = Event_Log.Program_ID ORDER BY Client_Names.Client_Name;.

4.5.2.2 Individual User History Report

This report shows the entire event history for a single selected user. The user is selected from a list box on the report properties form (See Figure 5. page 15). The SQL statement below populates this report. SELECT Event_Log.Event_Start_Time, Client_Names.Client_Name, Program_Names.Program_Name FROM Program_Names INNER JOIN (Client_Names INNER JOIN Event_Log ON Client_Names.Client_ID=

Event_Log.Client_ID) ON Program_Names.Program_ID = Event_Log.Program_ID WHERE Client_Name = sUserName

At the end of the SQL statement is a variable which acts as a parameter for the query. This variable is populated by the selection in the client names list box (see figure 5).

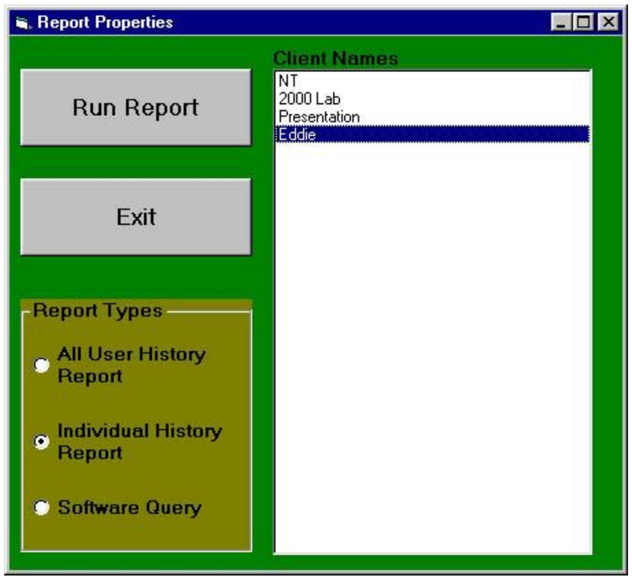
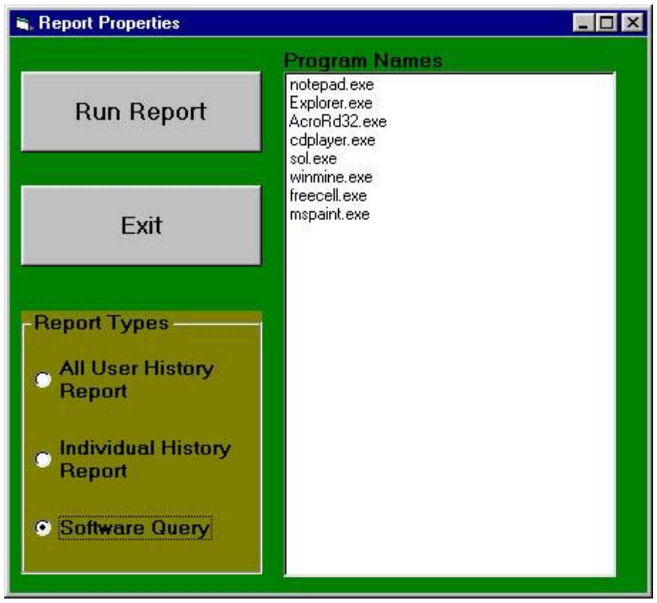


Figure 5. Client Names List Box
(Screen shot showing Client Names List Box)

4.5.2.3 Software Usage Report

This report generates a list of users who have used a selected program. The programs are chosen from a list box on the report properties form (see figure 6. page 16). This report is populated by the following SQL statement: SELECT DISTINCT Client_Names.Client_Name, Program_Names.Program_Name FROM Program_Names INNER JOIN (Client_Names INNER JOIN Event_Log ON Client_Names.Client_ID = Event_Log.Client_ID) ON Program_Names.Program_ID = Event_Log.Program_ID WHERE (((Program_Names.Program_Name) =

sProgramName)). The variable "sProgramName" at the end of the previous SQL statement is used as a parameter which is populated from the selection in the Program list box.



notepad.exe
Explorer.exe
AcroRd32.exe
cdplayer.exe
sol.exe
winmine.exe
freecell.exe
mspaint.exe

Figure 6. Program Names List Box
(Screen shot showing Program Names List Box)

4.6 Deliverable #6

4.6.1 Definition

A timer set for twenty seconds for the collection or evaluation of new events.

4.6.2 Description

Using the timer control provided by Visual Basic I set the interval property equal to 20,000. The timer control's interval property counts in milliseconds, so to get 20 seconds I set its value to 20,000. Twenty was chosen because it allows for better detection of programs they are actually spending time on, not just all programs accessed.

4.7 Deliverable #7

4.7.1 Definition

Create an Installation package for the client application.

4.7.2 Description

Using the Visual Basic Packaging wizard I created an installation program which loads all needed ocx files into the windows registry. This also installs the necessary dlls if they are not present on the system before installation. The following is a list of all of the files included in the packaged installation of the client: comdlg32.ocx, mswinsck.ocx, The Client executable, Psapi.dll, setup.exe, setup1.exe, stunst.exe, VB6 Runtime & OLE Automation, and VB6stkit.dll.

4.8 Deliverable #8

4.8.1 Definition

Local storage of Client event data until server makes a Winsock connection and collects events.

4.8.2 Description

Local storage of event data on the client machine is done using a text file. As each new event is detected a new line is created in the local text file. The events are added to the text file using a sequential access method. The file is opened using the OPEN command with the APPEND attribute. The PRINT command is then used to load the event variable into the next open line of the text file. Then the text file is closed until a new event is detected. For an example of a text file created by this code see **Appendix F**.

4.9 Deliverable #9

4.9.1 Definition

Create an Installation Package for the server application.

4.9.2 Description

This was created using the Microsoft Packaging Wizard. It loads all files for needed the program's operation into the registry and appropriate folders. If ODBC drivers are not present it will also create add them to the system. For a complete list of this install packages contents see **Appendix G**.

5. Conclusions and Recommendations

5.1 Conclusions

This project taught me a lot about the use of the Windows API. Before embarking on my senior project I had never used an API call. I had to learn how to use them and search through seemingly endless unorganized sets of documentation on the windows operating system architecture. By doing this I learned how the operation system works on a much more detailed level. I also gained a lot more information on programming procedure. I learned how to create classes to modularize my code into organized units based on functionality. This was a new concept. In the past all of my coding was composed of long lists of instructions which were difficult to debug. By modularizing I learned a lot about what Object Oriented Programming is all about.

Project management was another large lesson for me. I learned that scheduling my time and following a procedure for coding is very important. I also realized how important the design phase of a project is. When I started I just figured that I would sit down and start coding. That very first day I learned a hard lesson. With a large programming assignment there really isn’t a definite starting point. Every task has to be broken down into sub-tasks and those tasks have to be carefully planned out. This part of project management was actually what helped me understand what modularizing code is all about and gave me a ‘big picture’ view of object oriented programming. I learned that any project can have great aspirations, however when it comes down to time limits there are always going to be limitations to what one can do. Originally I had much larger ideas for this project, I learned that when taking on a project individually the workload is far greater. The next section describes all of the advances I would like to make to my software in the future.

5.2 Future recommendations

5.2.1 DHCP support

Use a different networking platform to account for networks which use DHCP in order to appeal to a greater customer base and cut down on program administration time. This would also allow the software to be run on larger networks.

5.2.2 Client Auto detect

This improvement would also cut down on software administration. It would also allow for real time information to be presented on the server side and eliminate the need for manual collection of client data.

5.2.3 Dynamic Timer Interval

Allow the event sample interval to be configured from the server to better suit the needs of each individual company.

5.2.4 Local Storage of Client Data - Registry

This would make the local data more secure during the time the client and server are not connected.

5.2.5 Calculate Each Event End Time

By calculating the end times of each event new reports based on duration of events can be created. These reports could be even more valuable to management.

5.2.6 Use of Charts on Reports

With the use of charts and graphs such as pie and bar PAT reports would become much easier to read by a larger group of people. It would increase the effectiveness of the data being presented.

5.2.7 Ranking System

A ranking system for could be developed to rate the value of each program being used. If this were to be implemented reports which showed the productivity of a person could be generalized as productive or unproductive with, less time spent on analysis of the data.

Appendix A
Timeline

STAGE 1: March 28th – May 23rd

1.1 Weeks 1-4: March 28th – April 18th

Develop Idea for project and write Problem/Area of inquiry report. Research topic and find possible sources. Meet with Russ McMahon and discuss Project details.

1.2 Week 5: April 19th – 25th

Speak with Russ McMahon and submit progress report 1.

1.3 Week 6-7: April 26th – May 9th

Interview Edward Hamann Sr., a small business owner. Write proposal first draft and meet with Shannon McClintock for approval.

1.4 Week 8: May 10th – 16th

Meet with Shannon McClintock for progress report 2 and talk with practice group about the presentations.

1.5 Week: 9 May 16th – 23rd

Meet with Randall Diekmeyer, a Systems Manager for interview. Finish final draft of proposal and turn in to Dr. Geonetta. Meet with practice group. Create presentation and deliver to audience on the 23rd.

STAGE 2: January 3rd – March 7th

2.1 Week 1: January 3rd – 10th

Format all hard drives and install software on all of the test machines. Set up peer-to-peer network.

2.2 Week 2-6: January 11th – 31st

Researched win32 API calls and learned how to use them using Visual basic. Began work on the client side application. Wrote pseudocode for program components. Met with Shannon and discussed progress.

2.3 Week 7-9: February 1st – 14th

Researched networking platforms. Continued client side development, designed and created database. Met with Shannon McClintock and turned in rough draft of design freeze paper.

2.4 Week 8: February 15th – 21st

Finished client side application for Windows NT OS. Completed server code and worked on networking. Met with Shannon McClintock to show functionality of the software.

2.5 Week 9: February 22nd – 28th

Completed server application/networking code. Added a rough display of report printout to show data access by the server for proof of concept.

2.6 Week 10: February 29th – March 7th

Finalized draft of design freeze paper. Spent time debugging software. Prepared presentation and present proof of concept.

STAGE 3: March 25th – May 31st

3.1 Month 1: March 25th – April 25th

Finish writing code and debugging project. Write code to make client compatible with Windows 9x. Create report formats. Clean up user interface.

3.2 Month 2: April 26th – May 22nd

Testing. Work on parsing issues for the program names. Organize Documentation. Package software and burn to CD. Work on final paper and presentation.

3.3 Month 3: May 23rd – May 31st

Finished writing Paper and prepared final presentation.

Appendix B
Budget

Software:

MS Visual Studio 6.0…………………………	$277.00	compuview-ms.com
1 License Windows NT………………………	$ 62.62	directdeals.com
1 License Windows 95………………………	$ 26.50	volumehardware.com
2 Licenses Windows 98…………...$56.62 * 2	$113.24	directdeals.com
Software Total:	$479.36	

Hardware:

4 15 inch Monitors……..……………127 * 4	$ 508.00	peachpc.com
4 Computers…..……………………$599 * 4	$2396.00	powerspec.com
4 Network Cards……………………………..	$ 24.00	computergeeks.com
1 10baseT Hub……………………………….	$ 14.00	www.ily.com
50' of Twisted Pair Cat-5 cable..…………….	$ 53.88	www.staples.com
Hardware Total:	$2995.88	
Total Budget:…………………………………	**$3475.24**	

I already own most of this material so not all of these materials will need to be purchased. The materials I already own are:

Software:
MS Visual Studio
2 Licenses Windows 98
1 License Windows 95

Hardware:
4 15" Monitors
1 Powerspec 366mhz PC
2 Micron P133mhz PC
1 P200 PC
4 3COM 3c509 NICs
50' Cat-5 twisted pair cable

With the components already on hand my total costs for this project will be:

1 License Windows NT………………………	$ 62.62	directdeals.com
One 5-port 10baseT Hub..……………………	$ 14.00	www.ily.com
Total Personal Expenses:……..……………	**$ 76.62**	

Appendix C

System Hardware and Software Requirements

This software is supported on all Windows 9x and NT or greater operating systems and hardware which support the OSs. The reason MS OSs were chosen is simply because they are the market leader in business software which allows for a larger market of my product as well. For appropriate functionality you will need a minimum of two computers networked with TCP/IP. The server application will need a licensed copy of Access 2000 installed. ODBC database drivers are also required for the database portion of this software to function. The server will provide them upon installation if they are not present on the system already. There is a minimum of 24 megabytes of space required for the Server installation and a minimum of 5 megabytes of space for the Client installation.

Appendix D
Windows 9x API Code

Class That Collects New Events

```
'Declarations for API calls
Private Declare Function GetForegroundWindow Lib "user32" () As Long
Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hWnd
          As Long, ByRef lProcessID As Long) As Long
Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As
          Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long)
Private Declare Function CreateToolhelp32Snapshot Lib "kernel32.dll" (ByVal
          dwFlags As Long, ByVal th32ProcessID As Long) As Long
Private Const TH32CS_SNAPPROCESS = &H2
Private Type PROCESSENTRY32
   dwSize As Long
   cntUsage As Long
   th32ProcessID As Long
   th32DefaultHeapID As Long
   th32ModuleID As Long
   cntThreads As Long
   th32ParentProcessID As Long
   pcPriClassBase As Long
   dwFlags As Long
   szExeFile As String * 260
End Type
Private Declare Function Process32First Lib "kernel32.dll" (ByVal hSnapshot As
          Long, lppe As PROCESSENTRY32) As Long
Private Declare Function Process32Next Lib "kernel32.dll" (ByVal hSnapshot As
          Long, lppe As PROCESSENTRY32) As Long
Private Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long)
          As Long
Dim iStartingPoint As Integer

Const SYNCHRONIZE              As Long = (&H100000)
Const STANDARD_RIGHTS_REQUIRED  As Long = (&HF0000)
Const PROCESS_VM_READ          As Long = (&H10)
Const PROCESS_VM_WRITE         As Long = (&H20)
Const PROCESS_DUP_HANDLE       As Long = (&H40)
Const PROCESS_QUERY_INFORMATION As Long = (&H400)
Const PROCESS_ALL_ACCESS As Long = (STANDARD_RIGHTS_REQUIRED Or
                                            SYNCHRONIZE Or &HFFF)
Const MAX_PATH = 260
```

Appendix D
Continued…

' This sub routine finds executable name of app running foreground window and ‘ tells whether it is different than the last time this function was called

```
Public Sub GetNewEvent(ByRef EventName As String, ByRef bIsNewEvent As
          Boolean, ByRef m_OldEventName)

   Dim lCurrenthWnd            As Long
   Dim lCurrentProcessID       As Long
  'Dim sCurrentModuleFileNamePath As String
   Dim sCurrentFileName        As String
   Dim sCurrentParsedFileName  As String

   'Get hWnd using the GetForegroundWindow API Call
   lCurrenthWnd = GethWndfromForegroundwindow

   'get ProcessID
   lCurrentProcessID = ProcIDFromhWnd(lCurrenthWnd)

   sCurrentFileName = ReturnFileName(lCurrentProcessID)

   'cuts extraneous spaces from filename
   sCurrentParsedFileName = ParseFileName(sCurrentFileName)

   If sCurrentParsedFileName = "" Then
      Exit Sub
   Else
      EventName = sCurrentParsedFileName
   End If

   bIsNewEvent = CompareWithOldEvent(sCurrentParsedFileName,
         m_OldEventName) 'detects wether is an old event

End Sub
```

'This function detects current foreground window and returns hWnd

```
Private Function GethWndfromForegroundwindow() As Long

  Dim lCurhWnd As Long
  lCurhWnd = GetForegroundWindow ’API call to get the hWnd of current
         ‘window
   GethWndfromForegroundwindow = lCurhWnd        'Return the hWnd

End Function
```

Appendix D
Continued…

```
'This function takes an hWnd and returns a ProcessID as a long
Private Function ProcIDFromhWnd(ByVal hWnd As Long) As Long

   Dim lThreadID As Long, lProcessID As Long
   lThreadID = GetWindowThreadProcessId(hWnd, lProcessID)
   ProcIDFromhWnd = lProcessID

End Function

' This function creates a snapshot of the current processes and then
' finds the filename of the active foreground task
Private Function ReturnFileName(ByVal lCurrentProcessID As Long) As String

   Dim hSnapshot As Long  ' handle to the snapshot of the process list
   Dim processInfo As PROCESSENTRY32  ' information about a process in that list
   Dim success As Long    ' success of having gotten info on another process
   Dim exeName As String  ' filename of the process
   Dim retval As Long     ' generic return value

   ' First, make a snapshot of the current process list.
   hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)

   ' Get information about the first process in the list.
   processInfo.dwSize = Len(processInfo)
   success = Process32First(hSnapshot, processInfo)
    ‘ Loop for each process on the list.
   Do Until sucess <> 0
     If processInfo.th32ProcessID = lCurrentProcessID Then
        ' Extract the filename of the process (i.e., remove the empty space)
        exeName=Left(processInfo.szExeFile,InStr(processInfo.szExeFile,vbNullChar)-1)
         ' Display the process name and the number of threads it owns.
         ReturnFileName = exeName
         Exit Do
      End If
         ' Get information about the next process, if there is one.
         processInfo.dwSize = Len(processInfo)
         success = Process32Next(hSnapshot, processInfo)
   Loop
    ' Destroy the snapshot, now that it isn’t needed.
   retval = CloseHandle(hSnapshot)
End Function
```

Appendix D
Continued…

'This function cuts path information from the string

```
Private Function ParseFileName(ByVal sCurrentFileName As String) As String

   Dim iLastCharLocation As Integer    'Location of last letter in program name
   Dim iFirstCharLocation As Integer   'Location of first letter in program name
   Dim iProgramNameLength As Integer   'Length of program name
   Dim sParsedFileName As String

   iFirstCharLocation = ((InStrRev(sCurrentFileName, "\")))
   iLastCharLocation = (InStrRev(UCase(sCurrentFileName), "E"))
   iProgramNameLength = (iLastCharLocation - iFirstCharLocation)
   If iProgramNameLength > 0 Then
      sParsedFileName = Mid(sCurrentFileName, (iFirstCharLocation + 1),
                              iProgramNameLength)
      ParseFileName = sParsedFileName
   Else
      Exit Function
   End If

End Function
```

‘ This function determines whether the event is new or not

```
Private Function CompareWithOldEvent(ByVal FileName As String, ByVal
         m_OldEventName As String) As Boolean ', ByRef EventName As String) As
         Boolean

   If FileName <> m_OldEventName Then
      CompareWithOldEvent = True
   Else
      CompareWithOldEvent = False
   End If

End Function
```

Appendix E
Windows NT/2K API Code

Class Which Collects New Events

There are only a few differences between the compatibility of the 9x client and the Nt/2K client applications. The class from appendix D holds the only code which differs from each version of the client. This section holds the code from the same class for Windows NT compatibility.

```
'Declarations for API calls
Private Declare Function GetForegroundWindow Lib "user32" () As Long
Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hWnd As
        Long, ByRef lProcessID As Long) As Long
Private Declare Function OpenProcess Lib "kernel32" (ByVal dwDesiredAccess As
        Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
Private Declare Function EnumProcessModules Lib "PSAPI" (ByVal hProcess As Long,
        ByRef hModule As Long, ByVal cb As Long, ByRef cbNeeded As Long) As
        Long
Private Declare Function GetModuleBaseName Lib "psapi.dll" Alias
        "GetModuleBaseNameA" (ByVal hProcess As Long, ByVal hModule As Long,
        ByVal lpBaseName As String, ByVal nSize As Long) As Long
Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As
        Long
Dim iStartingPoint As Integer

' Declaration of Constants
Const SYNCHRONIZE            As Long = (&H100000)
Const STANDARD_RIGHTS_REQUIRED  As Long = (&HF0000)
Const PROCESS_VM_READ          As Long = (&H10)
Const PROCESS_VM_WRITE         As Long = (&H20)
Const PROCESS_DUP_HANDLE       As Long = (&H40)
Const PROCESS_SET_INFORMATION   As Long = (&H200)
Const PROCESS_QUERY_INFORMATION As Long = (&H400)
Const PROCESS_ALL_ACCESS       As Long = (STANDARD_RIGHTS_REQUIRED
                                          Or SYNCHRONIZE Or &HFFF)
Const MAX_PATH = 260
```

Appendix E
Continued…

'* This sub routine finds executable name of app running foreground window and
'* tells whether it is different than the last time this function was called.

```
Public Sub GetNewEvent(ByRef EventName As String, ByRef bIsNewEvent As
Boolean, ByRef m_OldEventName)

   Dim lCurrenthWnd           As Long
   Dim lCurrentProcessID       As Long
   Dim lCurrentProcessHandle   As Long
   Dim lCurrentModuleHandle    As Long
   Dim sCurrentFileName        As String

   'Get hWnd
   lCurrenthWnd = GethWndfromForegroundwindow

   'get ProcessID
   lCurrentProcessID = ProcIDFromhWnd(lCurrenthWnd)

   'get Process Handle
   lCurrentProcessHandle = ProcHandleFromProcessID(lCurrentProcessID)

   'get module handle
   lCurrentModuleHandle = ModuleHandlefromProcessHandle(lCurrentProcessHandle)

   'get file name
   sCurrentFileName=ModuleFileName(lCurrentProcessHandle, lCurrentModuleHandle)

   'Close Process Handle
   CloseHandle lCurrentProcessHandle

   'Close Module Handle
   CloseHandle lCurrentModuleHandle

   'cuts extraneous spaces from filename
   'sCurrentParsedFileName = ParseFileName(sCurrentFileName)

   If sCurrentFileName = "" Then
      Exit Sub
   Else
      EventName = sCurrentFileName
   End If
      'detects whether it is an old event
   bIsNewEvent = CompareWithOldEvent(sCurrentFileName, m_OldEventName)
End Sub
```

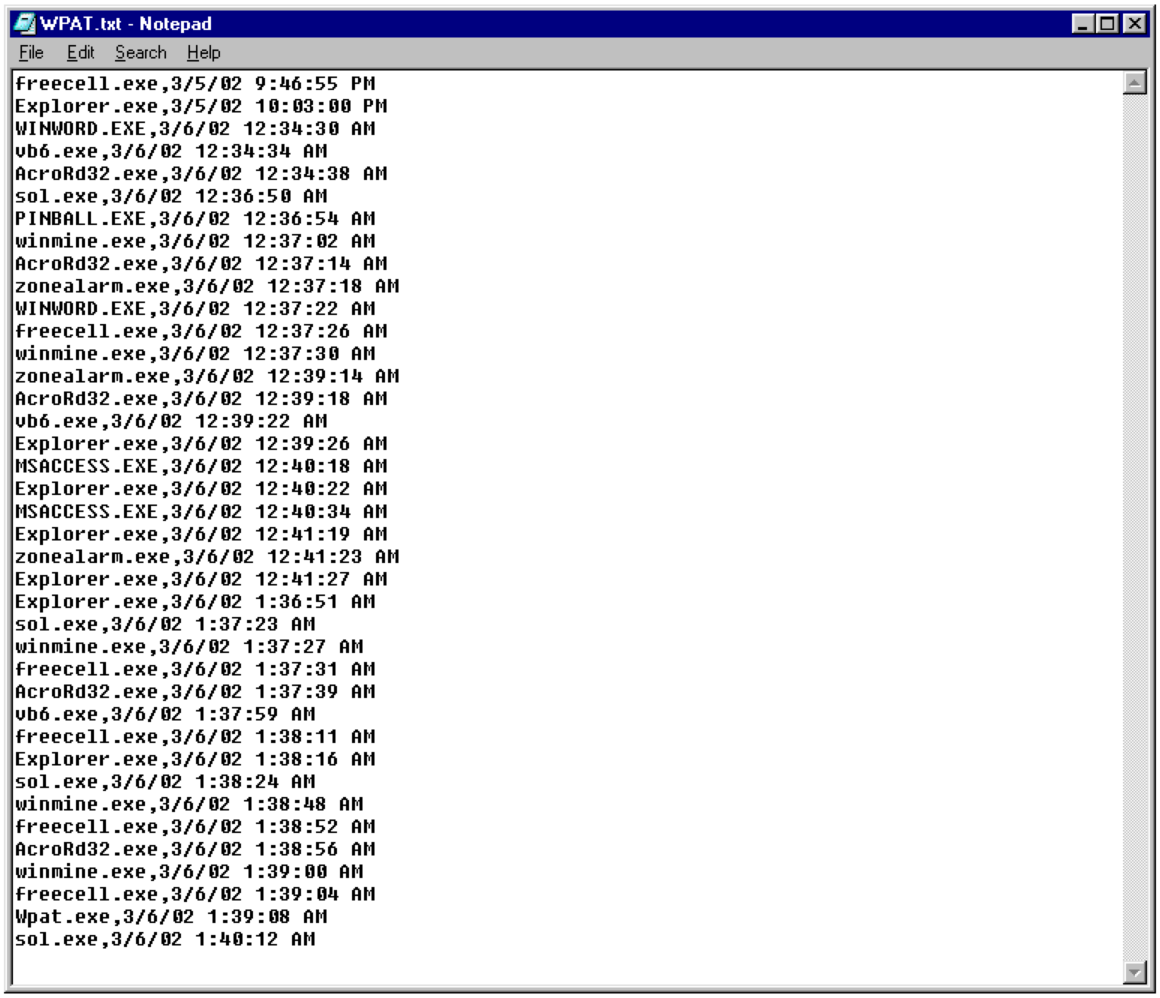
Appendix E
Continued…

```
'* Detects current foreground window and returns hWnd
Private Function GethWndfromForegroundwindow() As Long

   Dim lCurhWnd As Long
   lCurhWnd = GetForegroundWindow      'API call to get the hWnd of current window
   GethWndfromForegroundwindow = lCurhWnd        'Return the hWnd

End Function

'* Takes an hWnd and returns a ProcessID as a long
Private Function ProcIDFromhWnd(ByVal hWnd As Long) As Long

   Dim lThreadID As Long, lProcessID As Long
   lThreadID = GetWindowThreadProcessId(hWnd, lProcessID)
   ProcIDFromhWnd = lProcessID

End Function

'* This Function will Get the Process Handle from the ProcessID
Private Function ProcHandleFromProcessID(ByVal lProcessID As Long) As Long

   Const nAccessRights As Long = PROCESS_QUERY_INFORMATION Or
         PROCESS_VM_READ
   'Don't forget to close the handle!
   ProcHandleFromProcessID = OpenProcess(nAccessRights, 0, lProcessID)

End Function

'* This function retrieves the module handle using the process handle
Private Function ModuleHandlefromProcessHandle(ByVal ProcHandle As Long) As
         Long

   ‘this really is a bool.  0 = false. everything else is true
   Dim bEnumProcModulesSucceeded   As Long
   Dim hModule                     As Long
   Dim nSizeOfHMODUE
   Dim cbNeeded                    As Long

   bEnumProcModulesSucceeded = EnumProcessModules(ProcHandle, hModule, 4&,
         cbNeeded)
   ModuleHandlefromProcessHandle = hModule

End Function
```

Appendix E
Continued…

```
'* This actually retrieves the filename using the module handle and the process
'* handle.
Private Function ModuleFileName(ByVal ProcHandle As Long, ByVal ModuleHandle
        As Long) As String

   Dim bGetModuleFilenameSucceeded As Long
   Dim sFileName                   As String

   sFileName = String$(MAX_PATH, 0)       ' sets length of string/file name buffer size
   bGetModuleFilenameSucceeded = GetModuleBaseName(ProcHandle, ModuleHandle,
        sFileName, Len(sFileName))
   If (bGetModuleFilenameSucceeded <> 0) Then
      ModuleFileName = sFileName
   Else
      Dim nLastAPICallErrorDetails As Long
      ' Use this to look up the standard windows errors
      nLastAPICallErrorDetails = Err.LastDllError
   End If

End Function

'* This function determines whether the detected event is new or not
Private Function CompareWithOldEvent(ByVal FileName As String, ByVal
        m_OldEventName As String) As Boolean ', ByRef EventName As String) As
        Boolean

   If FileName <> m_OldEventName Then
      CompareWithOldEvent = True
   Else
      CompareWithOldEvent = False
   End If

End Function
```

Appendix F
Sample of Text File Created by the Client for Local Storage



```
freecell.exe,3/5/02 9:46:55 PM
Explorer.exe,3/5/02 10:03:00 PM
WINWORD.EXE,3/6/02 12:34:30 AM
vb6.exe,3/6/02 12:34:34 AM
AcroRd32.exe,3/6/02 12:34:38 AM
sol.exe,3/6/02 12:36:50 AM
PINBALL.EXE,3/6/02 12:36:54 AM
winmine.exe,3/6/02 12:37:02 AM
AcroRd32.exe,3/6/02 12:37:14 AM
zonealarm.exe,3/6/02 12:37:18 AM
WINWORD.EXE,3/6/02 12:37:22 AM
freecell.exe,3/6/02 12:37:26 AM
winmine.exe,3/6/02 12:37:30 AM
zonealarm.exe,3/6/02 12:39:14 AM
AcroRd32.exe,3/6/02 12:39:18 AM
vb6.exe,3/6/02 12:39:22 AM
Explorer.exe,3/6/02 12:39:26 AM
MSACCESS.EXE,3/6/02 12:40:18 AM
Explorer.exe,3/6/02 12:40:22 AM
MSACCESS.EXE,3/6/02 12:40:34 AM
Explorer.exe,3/6/02 12:41:19 AM
zonealarm.exe,3/6/02 12:41:23 AM
Explorer.exe,3/6/02 12:41:27 AM
Explorer.exe,3/6/02 1:36:51 AM
sol.exe,3/6/02 1:37:23 AM
winmine.exe,3/6/02 1:37:27 AM
freecell.exe,3/6/02 1:37:31 AM
AcroRd32.exe,3/6/02 1:37:39 AM
vb6.exe,3/6/02 1:37:59 AM
freecell.exe,3/6/02 1:38:11 AM
Explorer.exe,3/6/02 1:38:16 AM
sol.exe,3/6/02 1:38:24 AM
winmine.exe,3/6/02 1:38:48 AM
freecell.exe,3/6/02 1:38:52 AM
AcroRd32.exe,3/6/02 1:38:56 AM
winmine.exe,3/6/02 1:39:00 AM
freecell.exe,3/6/02 1:39:04 AM
Wpat.exe,3/6/02 1:39:08 AM
sol.exe,3/6/02 1:40:12 AM
```

Appendix G
Contents of Server Installation Package

Below is a list of all of the files used by the installation package to install the server side of the application.

Comdlg32.ocx
DAO350.dll
Expsrv.dll
MDAC_Typ.exe
MSADODC.ocx
MSBind.dll
MSDatalist.ocx
MSDBRPTR.dll
MSDerRun.dll
MSJet35.dll
MSJint32.dll
MSJTER35.dll
MSRD2X35.dll
MSRDO20.dll
MSREPL35.dll
MSSTFMT.dll
MSVCRT40.dll
MSWinsock.ocx
ODBCTOOL.dll
RDOCURS.dll
Setup.exe
Setup1.exe
St6Unst.exe
VB5DB.dll
VB6 Runtime & OLE Automation
VB6STKIT.dll
VBAJET32.dll
WPAT.exe
WPAT.mdb

References

1. "Bosses with X-Ray Eyes – Special Report on Electronic Privacy," *Macworld*, July, 1993

2. Conry-Murray, Andrew. "Special Report – The Pros And Cons Of Employee Surveillance". Network Magazine. February (2001)

3. Diekmeyer, Randall, Systems Manager. PEDCo E & A Services Inc. Personal Interview. 17 May 2001.

4. Dixon, Rod. "*With Nowhere to Hide: Workers are Scrambling for Privacy in the Digital Age"*, Journal of Technology Law & Policy, Issue 1, Volume 4, (1998-1999).

5. eNFILTRATOR Software, "Black Box Plus Alert Benefits", http://www.enfiltrator.com/black_box/index.htm

6. Further Innovations, "How Oculus Benefits your Business", http://www.furtherinnovations.com/OculusBenefits.pdf, 2001

7. Gahtan, Alan. "Monitoring Employee Communications". Web World, January (1997)

8. Hamann, Edward, CEO. Edward P Hamann and Associates Inc. Personal Interview. 3 May 2001.

9. Hillenbrand, Ann. "The Need to Know Versus the Need for Privacy". Radford University News. March, 8 (2000)

10. Krohn, Dan. "How far should employers go in monitoring their employees' electronic communications?". Huston Business Journal

11. Office of Technology Assessment, "The Electronic Supervisor: New Technology, New Tensions," 1987.

12. Portnoy, Jeffrey; Wang, Sarah O. "EMPLOYER ACCESS VS. EMPLOYEE PRIVACY: WHAT INFORMATION CAN EMPLOYERS OBTAIN?". http://www.cades.com/pubs/html/emp_law4b.html (1997)

13. Roman, Steven. Win32 API Programming with Visual Basic. O'Reilly, 2000