2010

ThumbScan: A lightweight thumbnail search tool

Joseph Elinski

ThumbScan

A lightweight thumbnail search tool

Joseph Elinski
jne2335@rit.edu
11/18/10

Yin Pan
Sumita Mishra
William Stackpole

Presented in accordance with the requirements for a Master's of Science in Networking and Systems Administration from

The B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology

Table of Contents

1. Introduction

1.1 Background

Since the introduction of Windows 95B and Windows 98, users of the Microsoft platform have been able to select a thumbnail view of any system folder [2]. By default, this view prompts the operating system to create a small thumbnail cache of all media in the chosen folder. This cache allows the operating system to quickly and easily retrieve each individual thumbnail data without needing to recalculate and redraw the miniature images each time that the folder is viewed.

The cache itself is stored as a hidden system file, named Thumbs.db or Thumbcache_*.db [10, 17]. Older iterations of the Windows operating system store this file in each individual folder. Newer versions, such as Visa and Windows 7, both store the thumbnail cache files in a single system folder.

The thumbnail cache utilizes Windows Object Linking and Embedding technology to create a custom storage format. When viewed in a Hex Editor, each database has the appearance of a small FAT file system. Inside, thumbnails are stored as separate data streams with standardized headers [9, 12]. File information, such as name and modification date, is then stored in a separate catalog stream.

1.2 Importance

Because of two major Microsoft design choices, thumbnail databases have become a useful aid in forensic investigation. First, Windows defaults to caching thumbnail images. This forces the operating system to create a thumbnail database for each folder that has been selected to show the thumbnail view. Though such an option can be disabled, there is little reason, beyond privacy concerns, for a user to do so. As a result, the thumbnail cache is commonly found on any eligible Windows system. Secondly, old thumbnails are not deleted from a given databases, even when the original media file has been removed from the system. Once a thumbnail is cached in a given database file, that thumbnail will remain there until overwritten or manually deleted. This enables old thumbnails to linger in a cache for extended periods of time.

By searching for and viewing the contents of thumbnail databases, a forensic investigator can easily examine the available media on a given system. Though the cache is not a perfect database of every stored image, it is a good indicator of media storage locations and habits. By examining the contents of every thumbnail cache on a system, an investigator can quickly locate contraband material. Additionally, if an investigator can locate illegal material stored within a thumbnail database, they can match the hash of that stored thumbnail to the hash of another Windows generated thumbnail of the original content. This allows them to prove that contraband material existed on a system, even if it has since been deleted [10]. Finally, if a user chooses to encrypt a number of files using the Encryption File System (EFS) option in Windows, then single files will remain encrypted, but their thumbnails may still be recoverable from the thumbnail database, as long as that is not encrypted as well [10]. In this way, and under the right circumstances, a skilled investigator can use this oversight to view the thumbnails of encrypted data, even though they cannot access the encrypted media directly.

2. Available Tools

No academic studies could be found on the forensic uses of the Windows Thumbnail Cache. However, a number of useful software packages and public documents were located.

2.1 Popular existing implementations

Current market products for locating and displaying the contents of the Windows Thumbnail Cache can be broken into three major categories: proprietary, professional-grade forensic suites; smaller, more focused proprietary tools; and free, open-source tools. There are very strict differences between the functions and scope of each.

Professional suites like the Forensic Toolkit (FTK) and Encase both offer functionality beyond simply reading a thumbnail cache. Instead, they are more focused on offering a full range of tools for forensic investigation. Thumbnail Cache investigation is simply a small feature within the entire forensic suite of tools. Additionally, they have been proven to operate without disrupting any data on a given image or drive. This is a function that all other tools did not openly guarantee. Of course, such extended functionality and proven operability comes with a heavy price; licenses to these programs are offered mainly at the enterprise level.

Encase contains the ability to make thumbnail caches searchable, by interpreting their internal container structure [7, 8]. This allows an investigator to view the contents of a cache, but only on a file-by-file basis. Recursive searching is not built-in for this purpose. As a result, locating and analyzing these files can be a long and arduous task.

FTK provides recursive search abilities, through the "List all descendants" option [1]. This allows an investigator to quickly and simply view all embedded thumbnails on a system. It does not, however, incorporate the ability to search unallocated space with this option. As a result, such file location is easier than Encase, but still not as simple and thorough as it could be. Our tool aims to solve these problems.

Encase and FTK are commonly accepted, court approved forensic suites. However, they are not the only private products available for forensic research. Several smaller companies, like GreenSpot Technologies, also exist for less formal investigations. GreenSpot has created a small suite of tools for image recovery and thumbnail cache viewing [5, 6]. These are offered with few publicly available details. It is known that they can recover images and perform low-level searches through the combination of offered tools. However, little information was available beyond that point. Documentation was scarce, and product downloads were dependant on preliminary payment. Our tool is free, open-source, and provided with detailed documentation into the functions of the executable.

Free, public tools, such as Vinetto, ThumbsUp, and the Thumbnail Database Viewer, all offer basic search and recovery functionality. Vinetto is a python-based tool created to aid *nix investigators in locating and extracting embedded thumbnails [15]. It runs natively on a Linux system, with the input of a single thumbnail database. It also offers search functionality, but only with the aid of piped output from the native Linux 'find' command [20]. Vinetto has been released in an alpha state, and is meant for testing purposes only. ThumbsUp is a java-based tool made to accept a thumbs.db file as input, with individual derived thumbnails as output. It is a good example of a java tool that incorporates the apache/Jakarta library to interpret the OLE structure of a thumbnail cache – many other, more amateur programs were located that utilized similar design considerations. ThumbsUp does not provide search capabilities, nor can it recover thumbnails from unallocated drive space. This severely limits its usefulness. Finally, the

Thumbnail Database Viewer is a small, simple tool to view individual thumbnail caches [18]. As with the other free tools, it does not offer search capabilities.

The located professional forensic suites offered search and extraction capabilities, but at a very high cost to the end-user. The discovered private tools provided similar functionality at a lower cost, but with little documentation and no universally proven abilities. Any similar free tools did not contain native search abilities, nor did they provide tool benchmarks and system impacts.

Our tool is encased in a free, open-source, stand-alone executable. It contains full recursive search functionality with the ability to recover thumbnail caches from unallocated space. Additionally, it offers advanced features report generation and an intuitive GUI for greater application control. In the end, it combines the best features of all available forensic tools, in respect to locating, cataloging, and extracting embedded thumbnails from the windows thumbnail cache.

2.2 Conclusions

Though popular tools currently exist for the analysis of the Windows thumbnail cache, no single program encompasses all of the necessary features for a quick, simple, and thorough search of an entire hard drive. Our implementation of a thumbnail cache search tool provides the proper functionality. This includes an automated, recursive search that is capable of traversing a standard file system and unallocated space in order to locate thumbnail cache files and extract any resulting embedded thumbnail images. All of this is controlled though an intuitive GUI that is executed from a single, portable binary file.

3. Research Methodology

3.1 Purpose

The purpose of this project was to create a small, lightweight thumbnail cache viewer that was capable of automatically and recursively pulling out every embedded thumbnail image stored in a given implementation of windows.

3.2 Program Objectives

This project aimed to create a search tool with the following functionality:

1. Recursive search capabilities
    a. Able to traverse an entire directory tree from a given start location
2. An intuitive and simple GUI
3. Above 95% accuracy in carving and displaying thumbnails
4. The ability to label thumbnails according to their original names
5. The ability to search unallocated space for deleted thumbnail caches
6. Full, automatic report generation with search statistics and findings
7. Encased in a small, stand-alone portable binary executable

3.3 Approach

This project was taken in an incremental approach. A base functionality was established, and then new features were added one-by-one until all desired goals were met. In this way, the project utilized the phase-gate method of project management.

In order to validate this thesis idea, thorough background research was performed. This included the creation of a small sample program in the Java language. Unfortunately, though simple and easy to use, Java does not offer the control necessary to operate on unallocated sectors of a Hard Drive. As such, a new programming language needed to be used. For this purpose, Visual C++ was chosen. This language had a very powerful backend capable of granting full disk control to the programmer. It also offered a clean GUI-creation tool and Microsoft-specific functions. As a result, the first phase focused on porting any current test code from Java over to Visual C++. Each subsequent phase in this project then introduced a new function to the base program.

The second phase focused on two simple features: GUI creation and file-naming. These were combined because of the relatively simple act of creating a functional GUI in Visual C++. Pulling out file names from the Thumbnail cache was a much more difficult task. This involved

analyzing the given cache for hex data that correlates each JFIF stream to a stored filename. Because these are known to appear in no particular order in the cache, further research and testing was done to determine a proper algorithm for labeling the extracted images. We knew that this feature was possible, as it exists in Encase and FTK. However, it was not implemented in the research version of this program, due to time constraints.

The third phase involved adding unallocated search functions into the program. This allowed the tool to automatically recover deleted and hidden thumbnail caches from a given drive. Such an attribute was essential to the overall impact and worth of this program.

Once all base features were added to the program, final testing and benchmarking were performed. This encompassed the final phase before a live demonstration and thesis defense. Each individual operation of the program was refined and optimized to ensure that the program remained small, portable, and efficient. They were also checked for forensic validity – proving that this executable did not modify any timestamps or other information during its operation.

As soon as the program had been proven to meet all goals, pass all testing, and record acceptable benchmarks, a final review and defense were scheduled.

4. Algorithm Theory

Before the initial program could be attempted, basic functions had to be planned in theory and in pseudo-code. These were separated into three separate functional categories: Search, Analyze, and Report.

This code was inaccurate and did not represent any single formal programming language. Instead, it acted as a base for what would become the final algorithms. It was a loose way of providing groundwork for how the functions would operate. As the actual code progressed, these were updated and modified to fit the given restrictions and libraries of the current language (Java or VC++).

4.1 Search

The largest differentiation between this forensic tool and others was going to be the built-in search function. By chance, this too became the simplest feature to implement. Using a standard recursive search function, the program was set to easily traverse a file system.

The initial search pseudocode was as follows:

```
Function search (directory dir){
    //get a list of all files in the given directory and save it to an array
    File[] files = dir.getFiles();

    //search that array for file types (directory vs file)
    for (int x = 0; x< files.length; x++){
        //if a directory is found, then search it using this function
        if ( files[x] == directory){
            search (files[x]);
        }
        //if a file is found, then check the name,
        //if it starts with thumb, then analyze as a thumbcache
        else if (files[x].startsWith("Thumb") ){
            analyze(files);
        }
    }
}
```

4.2 Analyze

The analyze feature was set to be the largest algorithm of the program. This function needed to retrieve the bit stream data from the file and then parse it for known information. In particular, it had to locate individual JPEG streams and save them byte by byte into a separate image file. These were found by searching for known byte-strings used only in JPEG files. Since most thumbnails were small enough to be held in one solid unfragmented stream, this was an easy process. It would not guarantee 100% accuracy, since it could not recover images in fragmented

streams, but it would work for most archived thumbnails. It would also prove to be small, quick, and efficient – three key features of the end product.

The pseudo code was as follows:

```
Function analyze (File tCache){
    //get byte data from file (this may require a loop if no getBytes type function is //available)
    StringBuilder sb = sb.append(tCache.getBytes());

    //loop through the byte data searching for the string JFIF
    //do this 1 byte at a time
    char testChar[] = {'a','b','c','d'};
    for (int x = 0; x < sb.length() - 3; x++){
        //grab four chars at a time to check for a match of "JFIF"
        testChar[0] = sb.charAt(x);
        testChar[1] = sb.charAt(x+1);
        testChar[2] = sb.charAt(x+2);
        testChar[3] = sb.charAt(x+3);

        //store the chars in one string
        String test = new String(testChar);

        //if you find a JFIF tag, start pulling out the associated picture
        if (test.equals("JFIF")){
            //write picture files until end of thumbnail is reached
            //start by creating the new file to hold the JPEG stream
            File pic = new File("Thumbs" + counter + ".jpeg");

            //increment the "pics found" counter this will number each file separately
            counter++;

            //create the output stream
```

```
            FileOutputStream out =  new FileOutputStream(pic);

            //create a counter to track the current byte
            int cByte = x+1;

            //while true, write data byte by byte to a new pic file
            //error check the data just before writing
            while (true){

                //write to the file until you reach the end
                if ( cByte < sb.length()){
                    //if you find the sequence 0xffd9, then you've found the end of the
                    //image stream
                    if ( sb[cByte – 1] == 0xff){
                        if (sb [cByte] == 0xd9){
                        out.write [cByte];
                        break;
                        }
                    }
                    //write the current byte to the image file
                    Out.write{sb.cByte];
                    //increment cByte to the next byte in the file
                    cByte++;
                }
            }
        }
    }
}
```

4.3 Report

A report function would generate the final log for the program. It needed to take data that had been saved from earlier in the program and then print it out into a formatted text file. Though this was not written out in pseudo-code as the other sections, it was still planned for. The initial Java Prototype had an Info class for this purpose, where file information was stored as an array of Info Objects, each holding name and path information. The final program, however, held this information as a single global array of strings.

5. JPEG Structure

Because all thumbnails are stored in the JPEG format, it was important to understand the basic structure and functions of this file format before attempting this project.

5.1 JPEG/JFIF Basics

The JPEG standard is a set of formats for digital image compression and storage [12]. The JPEG File Interchange Format (JFIF) is an addition to this, which defines rules for a common universal image format that can be shared between operating systems [9]. The JFIF standard outlines several storage aspects that were previously unspecified. In the end, it simplifies and streamlines the specification to an easily interpreted and transmitted format. Because of this, many modern operating systems and programs use this standard for creating and storing digital images.

5.2 Marker Codes

The JPEG format uses marker codes to identify specific sections and data structures within a file. Each code is two bytes long and starts with 0xFF [12]. These designate areas such as the Huffman table, quantization table, Start of Image (SOI), and End of Image (EOI).

Marker codes were especially important for this program. There are only two reliable ways to locate and extract embedded images from an OLE file. The first, and most complicated, is to interpret the entire structure of the file, thus following individual storage streams to locate an image. The easier, though less detailed, method is to search for the JPEG header or known JPEG byte strings. This was the search method of choice for this iteration of the program, due to its simplicity and efficiency.

In relation to this program, the four most important marker codes were as follows:

1. The Header -- FF D8 FF E0
2. The JFIF standard identifier – 4A 46 49 46
3. The Start of Image (SOI) -- FF D8
4. The End of Image (EOI) -- FF D9

These were used to locate and extract thumbnails with a given thumbnail cache.

6. Thumb Cache Structure

The internal structure of the Windows Thumbnail Cache is based on the Microsoft Open Linking and Embedding (OLE) format [3,15]. It follows the basic guidelines of the format but does not seem to strictly adhere to them.

6.1 OLE Basics

An OLE file is comprised of separate fragmented virtual streams of data [13, 14, 16]. Each stream is allocated in sectors and can be described as a management (Header or Directory) or storage (FAT, mini-FAT, or Double-Indirect FAT (DIF)) structure.

The easiest way to picture an OLE file is to start with the Header. This is the true root of the file. It is a parent to the Directory and DIF streams. These are then parents to an unlimited number of Storage streams. In this way, the entire file is a tree of nodes with parent-child relationships – much in the same fashion as a standard file system. This is why the OLE structure is commonly referred to as a modified FAT file system.

6.2 Management Sectors

The first sector of an OLE file (SID 0) is always the Header. Each OLE file will only have one 512 byte header at the beginning of the file. This stores key attributes of the file, such as the location and size of main sectors of the file.

The header is traditionally followed by a Directory sector, though this is not required. An OLE Directory always starts with a root element, which is designated by the “R” character. Depending

on the version of the OLE file, this can either be a standalone “R” or it can be the first character in the string “Root Entry”. The Directory contains an index of FAT and mini-FAT streams within the given file.

6.3 Data Sectors

OLE storage is comprised mainly of FAT streams. These are organized as chains of FAT sectors. They can be equated to a linked list of elements, where each contains information about the previous and subsequent elements in the chain.

Within a chain, elements are marked as either a stream or storage member. Streams must have a start sector and size. These describe the chain itself. Storage, on the other hand, must have a timestamp and information about child sectors. These hold the actual payload of the data element.

Three types of storage exist within an OLE file. The FAT is the standard form of storage. By default, FAT sectors are 512 bytes each, and are used to hold a payload of more than 4096 bytes. For smaller files, mini-FAT streams are used. Each mini-FAT sector is traditionally 64 bytes long. These are held within the mini-FAT stream, which is a subset of divided FAT sectors.

The first 109 FAT sectors of an OLE file are described by the Directory. Any subsequent sectors are represented by the DIF.

6.4 Discoveries

Through investigation and the attempted creation of an OLE interpretation algorithm, many details of the thumbnail cache and its manipulation of the OLE format were uncovered.

6.4.1 Endian Order

In regards to programming, one of the most important OLE discoveries was the abundant use of little-endian values within the structured files. Little-endian refers to a hexadecimal number recorded with the least-significant bit first. This is the opposite of standard numerical representation.

The majority of embedded file values, such as sector locations and recorded attributes, were provided in the little-endian format. As such, all interpreted values had to be reversed before being converted into a more standard format.

6.4.2 Unicode Use

All plain-text values within the OLE files are stored in Unicode format rather than the more standard ASCII. Because of this, all strings within each thumbnail cache had to be converted in order to be interpreted correctly.

6.4.3 Use of DIF

The DIF is only located in an OLE file that is over 7Mb in size. Because most thumbnail images were found to be a mere 4kb, it was estimated that a thumbnail cache would only contain Double Indirect FAT sectors if a single given folder held more than 1,792 thumbnails.

6.4.4 Stream use

Given the small size of individual thumbnails, the investigated OLE files were found to rely heavily on mini-FAT streams. This was to be expected. However, the original file names of these thumbnails were interspersed between these standard entries with no discernable structure.

As referenced below in Figure 6.4.4.1, you can see the EOI for an image (FF D9), followed by an 80 byte block that contains interspersed name data before an new block with an SOI marker is reached.

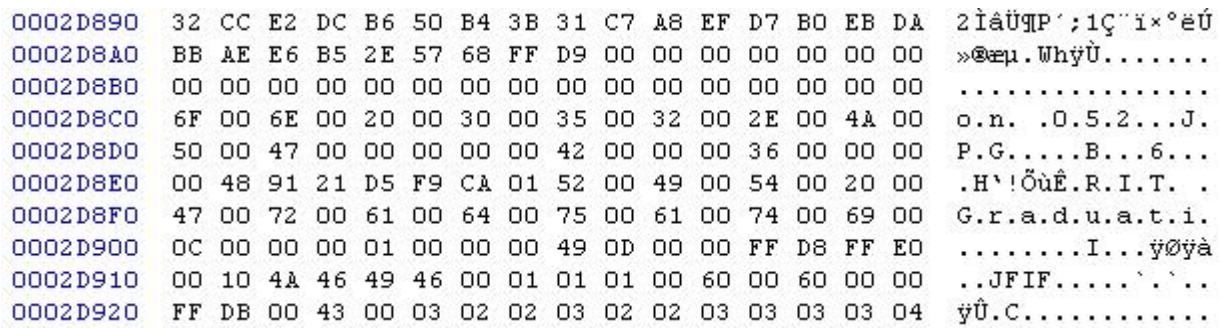

Figure 6.4.4.1 File Names within a Thumbnail Cache

In comparison, Figure 6.4.4.2 shows an EOI, followed by 64bytes of name data before a new block with an SOI marker.

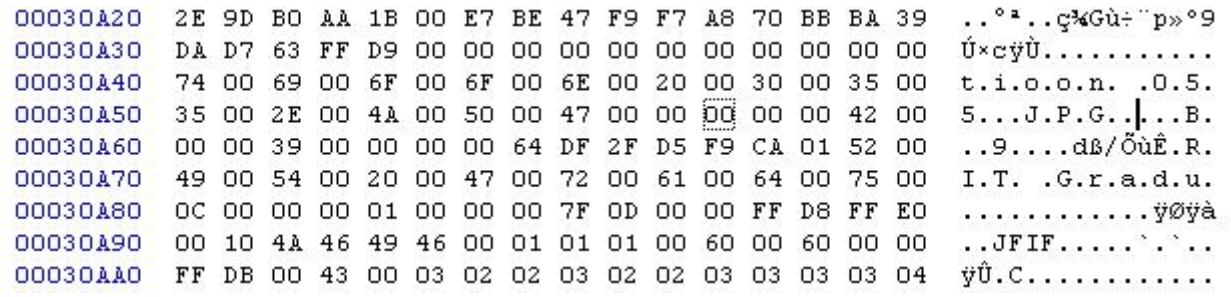

Figure 6.4.4.2 Alternate File Names within a Thumbnail Cache

Manually viewing individual thumbnail caches in a hexadecimal editor turned up dozens of such examples, where name data was placed in differing block lengths within the files. In addition, it was often presented out of order and in the Unicode format. This made identifying them difficult, even though they were in plain-text. At the time of this writing, a standard method for indentifying and interpreting this structure was not yet discovered.

7. Program

7.1 Prototype

In February of 2010, a prototype of the ThumbScan program was created. This program featured the base functions of the most recent program, and it acted as a proof of concept for the final design. This program was written in Java and compiled with the latest java release that was available at the time (1.6.0_18).

The prototype allowed a user to recursively scan a local system and carve JPEG streams from every discovered Thumbs.db file. It did so in a very primitive fashion, and therefore produced very simple results.

The program was primarily run using the command line, and it offered one single option – to designate the starting directory. Discovered images were placed in the running directory along with a text report that listed basic information about each file.

A standalone executable JAR version of the program was provided with the Scanner package. Users were warned that the executable started scanning an entire system starting from the root C:\ directory, and that it did so without output to a visible console - since Windows associates

javaw.exe with the jar extension. Once finished, it wrote all results to a log file in the directory that it was executed from.

The prototype ran using three supplied class files and a signatures file. They were as follows:

ThumbScan.class

This was the main program class. It managed calls to the FindFiles scanner and formatted the program output.

FindFiles.class

This class recursively searched the file system using a given start point. By default, it started from the root of the C: drive. It then searched through every discovered folder for files named “Thumbs.db”. It would load each discovered thumbnail cache into memory and search byte by byte for JPEG streams. These were identified by their magic number. Each JPEG stream was the carved out into a separate numbered image file, and the known information about the file was saved as an Info object.

Info.class

This class acts as a struct, an object that stores given types of data. It was used to keep track of file names, locations, and parent directories for every discovered thumbnail cache or JPEG stream.

This program was slow and resource intensive, but it paved the way for the final carving algorithms that were used in ThumbScan. It was a testing ground. Unfortunately, because of restrictions in the Java language, it could never be more than this. Advanced functionality was required. More so, the program needed to be faster and more efficient. This is why Visual C++ was chosen as the language for a new version of the program. The Microsoft-centric language provided the low level power and efficiency of C++, with added functions for building a clean GUI and interfacing directly with the Windows operating system.

7.2 ThumbScan Features

As outlined in the initial project specification, ThumbScan was designed to incorporate a number of key features that allowed it to stand out from other similar forensic tools.

7.2.1 Recursive Search

ThumbScan searches an entire directory tree using a standard recursive algorithm.

The search function retrieves the contents of a given folder. If files are found, then it scans the names for anything that matches a windows thumbnail cache and acts accordingly. If folders are located, then the search function is called again with each given folder. In this way, it quickly and efficiently scans a file system.

7.2.3 Simple GUI

The GUI provides users with a simple set of options and preset scan modes.

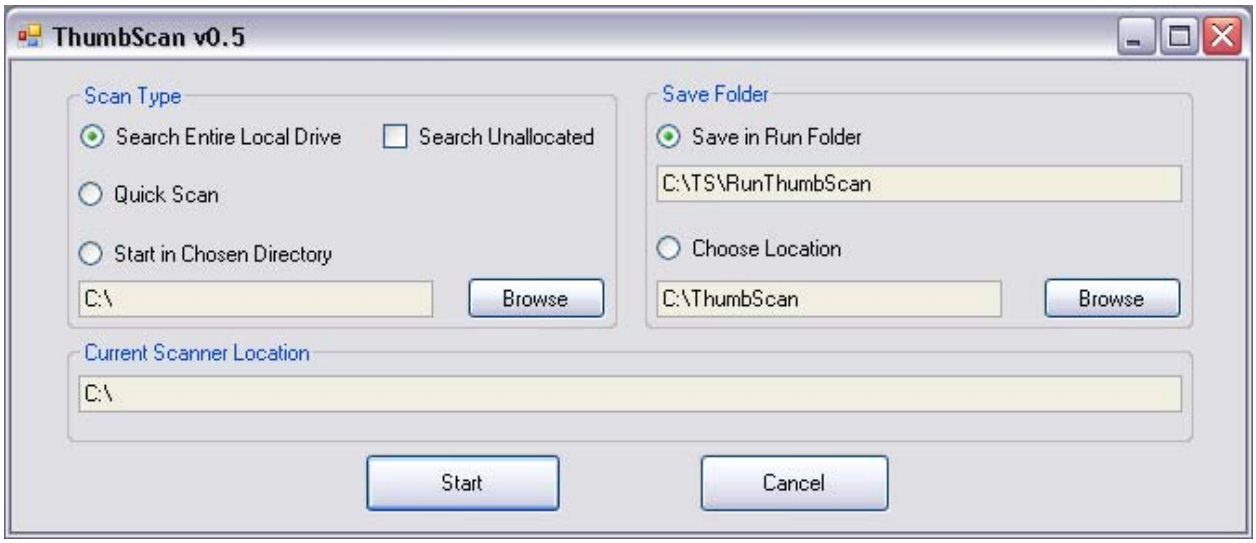

Figure 5.2.1 ThumbScan GUI

The user is provided with two main option categories: Scan Type and Save Folder. Scan Type determines where the program starts scanning from. The Default option is to search the entire C: drive starting at the root level. A quick scan only searches the Documents and Settings or Users folder, depending on the OS type. A user can also choose to start in a given directory. Additionally, a user can choose to search unallocated space as well. This option automatically scans the MFT and subsequently recovers and analyzes any located Thumbnail Caches that have been deleted. The Save Folder options simply denote where the scan results will be saved. This includes all recovered thumbnails, logs, and undeleted thumbnail caches. The default is to save in whatever folder the executable was launched from, though a user can also choose a custom location.

While running, the GUI will display the current folder in the “Current Scanner Location” field. This gives the user a sense of the scan status, since a progress bar would be impossible to calculate. When scanning unallocated space on the drive, this field simply lists $MFT as the current location.

7.2.3 Undelete

As an added option, users can choose to scan unallocated sections of a drive for deleted thumbnail caches. This function utilizes open source code from NTFSUndelete to scan the Master File Table of a Windows system for file entries with the wFlags bit not set – thus designating them as deleted. In this case, the file name is retrieved and compared to known names of the windows thumbnail cache. If a match is found, then the file is recovered and analyzed as normal. This allows a user to quickly scan a Windows file system for thumbnail caches that have recently been deleted.

7.2.4 Fast

By slimming down on unnecessary nested if statements, the code has been optimized. Compared to the Java prototype, the latest version of ThumbScan scans a system at blinding speeds. In simple tests, it was able to scan 150GB of data to reteive 14,700 thumbnails in just under 13.5 minutes.

7.2.5 Small

The current GUI-enabled version of ThumbScan is encased in a single 1Mb executable. This is a mobile executable, which does not include any kind of installation beyond common Windows libraries. As long as a system already has Visual C++ Redistributable 2010 and .Net Framework 4.x installed, this program will run.

7.3 Program Use

The current iteration of ThumbScan is best used for research-based applications. However, future versions could hold real-world forensic functionality.

7.3.1 Light scanning

ThumbScan provides quick, simple scan functions. It allows a user to determine the past and present media contents of a drive within minutes. For this reason, the current version is perfect

for casual use. As an example, spouses or employers could use it to locate questionable material on a given computer. However, because the program cannot yet pull out full file details, scan a system without modifying access times, or produce a professional report; it should not be used for legal purposes. It does not yet provide the level of sophistication required for such applications.

7.3.2 Potential Future Use

With some minor additions, this program could prove to be a formidable forensic tool. Most of these would come in the form of algorithm refinement and enhanced functionality. All such changes are outlined in Future Work section of this report.

With added development, this program could be used in a professional forum to extradite forensic investigation. In most cases, it is hypothesized that this tool would be used as a preliminary scan, directing investigators to key areas of interest in a given file system. It would cut down on manual search times by allowing investigators to quickly find hidden media on a system, whether or not it had been deleted. Then, other court-recognized tools could be used to create a viable legal case.

7.4 Code Structure

ThumbScan is written in VC++ using a standard object oriented method.

The entry point of the program is ThumbScan.cpp. This is a simple skeleton class that makes the initial call to Form1. The Form1 class contains all basic GUI creation and functionality. It creates the visible form, captures user option selections, and initiates the system scan with a button action. When directed by the user, it begins the scan by making a call to Scanner.cpp.

The Scanner class holds the main scan, search, and analyzation functions. This is the heart of ThumbScan. It takes the given user options and calls appropriate private classes that are named by primary function based on collection or analyzation. If an unallocated search was selected, then NTFSDrive.cpp and MFTRecord.cpp are called.

NTFSDrive and MFTRecord are modified versions of an open source program posted to codeproject.com by T.YogaRamanan. They are unmanaged MFC classes that read in the Master File Table of a given system. The classes then analyze the MFT entries for files that are indexed

have a wFlags value of 0 – meaning they have been deleted. ThumbScan uses a version of these classes that has been debugged and repurposed for the needs of our tool.

7.4.1 Fom1.h

The form1 header acts as a control for all basic GUI functions. It contains generated code for producing the user interface. All interactive fields and their prospective functions have also been defined within this class.

The most important section of this class is the startButton_Click method. This collects the user-set options, and creates a new instance of the Scanner class to analyze the given drive.

```
if (localSaveRadio->Checked){
    gcnew Scanner(currentFolder->Text, runFolderText->Text, currentFolder,
    unAllocatedBox->Checked);
}
else{
    gcnew Scanner(currentFolder->Text, saveFolderText->Text, currentFolder,
    unAllocatedBox->Checked);
}
```

Figure 7.4.1.1 startButton_Click method

The main variables for this function are the selected start folder (currentFolder), the save location runFolderText or saveFolderText depending on the chosen option), a pointer to the status bar object (currentFolder), and whether or not an unallocated search was chosen (unAllocatedBox).

7.4.2 ThumbScan.cpp

ThumbScan.cpp contains little more than the main function of the ThumbScan program.

```
int main(array<System::String ^> ^args){
    // Enabling Windows XP visual effects before any controls are created
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    // Create the main window and run it
    Application::Run(gcnew Form1());

    return 0;
}
```

Figure 7.4.2.1 Main Function

The main function acts as en entry point into the program. It calls for the creation of the GUI. All code in this section was generated by Microsoft Visual Studio 2010. No modification was necessary.

7.4.3 Scanner.cpp

The Scanner class is the heart of ThumbScan. It contains all basic functions of the program.

Run(): The Run function sets up the basic attributes of the Scanner class. It initiates a counter to time the length of the scan. This is followed by a call to the Collect function to start the scan and file analyzation. Finally, once the main portions of the Scanner class have completed, this function completes the final clean up by displaying the total scan runtime, writing all collected information to a log, and then opening the save folder in explorer.

***Collect()*:** The Collect function begins by grabbing a listing of every folder and file in the current given directory. Each folder is recursively searched, while the files are tested for a name match to “Thumbs.db” or “thumbcache”. If a match is found, then the file is sent to the Analyze function.

CollectDeleted(): The CollectDeleted function creates a handle to the Master File Table, then reads it in to memory. From there, the file is scanned entry by entry for deleted files. When one is located, it is tested for a name match to “Thumbs.db” or “thumbcache”. If a match is found, then the file is recovered to the save directory and sent to the Analyze function.

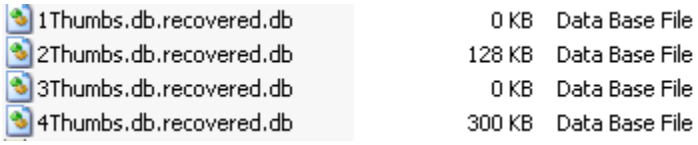

Figure 7.4.3.1 Databases Recovered by the CollectDeleted function

Analyze()*:* The Analyze function is the most complicated in the entire program. It reads every byte of the thumbcache into a StringBuffer. It then scans the StringBuffer byte by byte for a 4-charcter match to ‘JFIF’, the marker for a JPEG stream. When one is found, the function pulls it out into a numbered file in the given save directory.

While programming for this project, it was discovered that a number of the JPEG files had extra information in the header that was not inherently readable by Microsoft. It is unknown why Microsoft would include unreadable information in images that they generate. In total, seven common extra byte strings were identified within the headers. The Analyze function is setup to ignore these based on a known match to their start and end.

AnalyzeStructure(): The AnalyzeStructure function was abandoned due to time constraints. The main structure of the algorithm is written, but it was never fine tuned to work correctly. The currently available version of this function does not successfully pull images from a thumb cache. It is unknown at this time why.

This function starts by verifying the Magic Number for the thumb cache. It then reads in important values from the header. This includes the location and size of the FAT and mini-FAT blocks. It then attempts to read through the file sector by sector according to the Directory. Standard streams are analyzed for key values, such as file name and timestamp. Storage streams are extracted as individual JPEG files.

This function is very well commented, due to the complexity of OLE files and the operation that it is trying to perform. It was created with the help of available documentation on the OLE

format. Analyzing available Open Source code also aided in determining how such a file could possibly be read and extracted.

The future plans for this function are outlined in the Future Work section of this report. In addition, encountered problems are detailed in both the OLE file section and the Conclusion.

7.4.4 NTFSDrive and MFTRecord

These functions were borrowed from a tutorial on NTFS file recovery from codeproject.com. Though the author could not be reached directly to verify their license, a note was found that detailed his wishes for his code to be freely used for any purpose, provided his name and a link to his work were attached to references of the code. In that sense, it should be noted that these classes were originally written by T.YogaRamanan and posted freely to http://www.codeproject.com/KB/files/NTFSUndelete.aspx for public use. These classes were then modified to resolve minor bugs and some major issues with mixing managed and unmanaged code. They were also repurposed; since both were originally part of a larger GUI based undelete application.

These classes aid in reading and interpreting the Master File Table of a given drive. They also assist in recovering a deleted file once found. They were invaluable to this project for the custom structs associated with NTFS and MFT items.

In the Future Work section of this report, plans are outlined to remove these two classes, and instead create a single class that is written specifically for this program.

8. Benchmarks

Once the final research version of this ThumbScan was written, the program was tested on both Windows XP and Windows 7 based systems for speed and forensic validity. The following sections detail these test systems and the calculated program run time.

8.1 Windows XP

Two Windows XP systems were used for testing. The first was a 5 year old Compaq R4000 laptop. The second was an 8 year old HP Pavilion a245c. These were used due to both

availability and their status as standard home computers. Fast run times on systems as dated as these would prove the efficiency and power of this program.

Of the two scans, the laptop took the longest to run the full unallocated scan of the C: drive. However, in 16.2 minutes it was able to carve 14,700 standard thumbnails and recover 19 images from deleted databases. This is an average discovery of 18 embedded images a second. Due to the vast number of files recovered in this first scan, an accuracy measurement was not calculated.

The second scan was significantly faster, due to the lower count of media on the given desktop. However, no unallocated thumbnail caches were discovered. For this system, 48 of the 2,563 located images were corrupted. This proved 98.13% accuracy in thumbnail recovery.

System 1:

Configuration:

Compaq R4000

Processor: Single Core 2.2 GHz AMD Athon 64 4000+

RAM: 1.5 GB PC2700

Storage: 150 GB PATA4 HD with 8 MB cache

Standard Search:

Run Time: 13.5 minutes

Images Located: 14,700

Unallocated Search:

Run Time: 16.2 minutes

Images Located: 14,719

Recovered databases: 4

System 2:

Configuration:

Hewlett-Packard Pavilion a245c

Processor: Single Core 2.0 GHz AMD Athlon 64 2800+

RAM: 2.0 GB PC2700 RAM

Storage: 500 GB PATA4 HD with 8 MB cache

Standard Search:

Run Time: 0.92 minutes

Images Located: 2,563

Corrupted Images: 48

Unallocated Search:

Run Time: 3.35 minutes

Images Located: 2,563

Recovered databases: 0

8.2 Windows 7

A single Windows 7 system was used for benchmarking. This was a Dell mini 9 with Windows 7 Professional installed. The system was relatively young, and had been implanted with both Windows XP and Windows 7 style thumbnail caches. Because of the age and lack of storage, very few images were available on the system. This scan proved the speed and carving abilities of the program when faced with a newer file system.

Though the unallocated search did not discover any deleted databases, both scans recovered all discovered files without corruption. In addition, the program was able to recover files from both the thumbs.db and thumbcache* style files.

System Details:

Configuration:

Dell Mini 9

Processor: Single Core 1.8GHz Intel Atom

RAM: 2.0 GB PC5400

Storage: 32 GB SSD

Standard Search:

Run Time: 0.89 minutes

Images Located: 2,563

Corrupted Images: 48

Unallocated Search:

Run Time: 2.3 minutes

Images Located: 23

Recovered databases: 0

8.3 Forensic Validity

After each benchmark test, the given file systems were analyzed to prove or disprove the forensic validity of the tool. Once the scan had completed, variations of the DOS ‘dir’ command were run in a random set of folders in order to view the modify, access, and creation times of the scanned files.

Though ThumbScan did not change the creation or modification times of the files, it did have a heavy impact on the access times of all encountered files. Unfortunately, this means that this program cannot be considered a true forensic tool. As a result, in the current stage of development, ThumbScan would best be used for light casual scanning.

9. Future Work

The presented version of this program was created for educational purposes. It provided a background for study into the structure of the Windows Thumb Cache as well as opening a discussion on the benefits and uses of such a tool. In order to provide a real, practical program, enhancements would need to be made.

9.1 Extended functionality

The current iteration of ThumbScan is functional, but only at the most basic level. Several small additions could be made to dramatically increase the usability of the program.

GUI Updates: The current GUI has been created with the help of Microsoft Visual C++. Though this language allowed for the simple and quick creation of such a GUI, it adds significant size to the program. Additionally, the use of this language means that appropriate libraries must be installed in any Windows machine before Thumbscan can be run. In this case of this version, VC++ Redistributable 2010 and the .NET framework are required. Because these are not, by default, included in standard builds of Microsoft Windows, potential program applications are severely limited.

Instead, a simple custom interface should be created. Though this would be more time-consuming, it could be done in such a way to break the dependence on VC++, thus allowing for a more self-contained program.

Advanced Data Recovery: Currently, the program is able to carve JPEG streams from a thumbnail cache. However, it does not have the ability to also recover file data about these individual thumbnails. The data appears within the Thumbnail Cache. However, a working function was never completed for this purpose.

An algorithm, based in-part on the available Python code from Vinetto, was written to recover the name and modification data from the Thumbnail Cache. Unfortunately, due to time constraints and runtime errors, it was never completed.

OS fingerprinting: If the program were setup to automatically determine the OS and scan based on that data, the program could easily run as an automated job. It could also be built to make

scan suggestions to the user based on these results. This was attempted, but a functional method of resolving the OS from Windows 98 through Windows 7 was never found.

Image Read Capacity: Since forensic investigators commonly analyze disk images to preserve data integrity, ThumbScan would become immensely more valuable if it followed the same guidelines. This would be a seemingly simple addition, if it were implemented as a bit-wise search function.

The program could search through a given disk image for bit strings that matched the known header to a thumbnail cache. In this way, it would easily be able to locate and extract thumbnail data using the algorithms that are currently in place. Since it would do so without having to run on a native system, it would become a more viable tool for professional forensic investigation.

HTML report generation: Currently, reports are generated as a simple text file. This contains basic information about the images that were found. This works for simple statistical analysis and image location, but not for easily browsing thousands of images, since Windows has to re-generate a thumbnail for each discovered image. If an HTML report were generated instead, then a user could browse images and details at the same time, leading to faster and more comprehensible results.

Undelete: The current implemented undelete classes provide a proof of concept for this project, but are lacking the functionality necessary for a professional product. As a result, they should be removed in place of a single custom class. This should be written from the ground up to scan the unallocated sections of a drive byte by byte for matches to the known header for a thumbnail cache. It should be done using a low level search, instead of a quick scan of the Master File Table. This would be slower than the current method, but would provide more robust results. Such a class should also be written to accept any designated drive, in place of the current class which will only scan the MFT of the C drive. If given enough time, functionally for reading and recovering files from FAT formatted drives could also aid in the practicality of this program, by allowing investigators to search thumb drives and other removable media for deleted thumbnail caches.

Forensically Sound: Though the current iteration of ThumbScan does not change creation or modification times of any files touched, it does change access times. In order for this program to

be a true forensic tool, it will need to traverse and analyze a file system without modifying any data.

9.2 Optimization

The current version of the code is functional, but not as elegant as it could be. For size and speed enhancements, the program should be optimized according to advanced coding standards. As an example, this could include the removal of unnecessary variable use, the identification and removal of bottlenecks, the simplification of complex statements, and the addition of threading. Also, the code could be re-written in standard unmanaged C++, so as to remove the extra processing power required by the garbage collector. In the end, such enhancements would allow for a smaller, faster, less resource intensive program. As before, such work would make this program more viable as a professional forensic tool.

9.3 Bug Fixes

Though the current iteration of this program is useful for research purposes and light use, it does present a number of bug fixes. The majority of these have been caused by the mixture of managed (CLR) and unmanaged (MFC) code. Most notably, the program has, on occasion, thrown SEH exceptions due to objects that are directly referenced in memory, but have since been moved by the active garbage collector. In this way, the program attempts to read a specific location, but finds no defined information. Unfortunately, this type of error cannot be caught and is therefore left to crash the program. Efforts have been made to bypass such issues, but the only true solution at this point is to use one single style of C++, rather than two.

There are likely to be additional errors as well, but due to the limited availability of skilled, knowledgeable testers, it is unknown how many are present at this time.

10. Conclusions

Through the research, development, and application of this project, many important discoveries have come to light. These have answered necessary questions while creating a string of new ones. Hopefully, this work will prove to inspire future work and investigation into the use of the Window thumbnail cache for forensic investigation.

10.1 OLE Structure

As referenced previously, the OLE standard seems to have been loosely followed for the purposes of the windows thumbnail cache. It is unknown why Microsoft has chosen to do so.

In terms of general observations, there was a strong use of the mini-FAT with a lesser focus on the FAT and DIF, largely due to the small size of the thumbnails. There was also a mixed use of endianness for embedded values. Though this did not cause major issues, it did make reading the files challenging.

With respect to major differences from the written standard, the names and other associated file information was found to be stored in a seemingly random manner. Where they were expected to appear in standard data streams with timestamp information, these were instead found spread throughout the file in differing lengths. This made interpreting the file difficult. More future research will need to be done in order to successfully decipher the internal structure of these database files.

10.2 Theory and Development

This project was based on a simple theory. Since modern versions of Windows (XP and above) utilized the JFIF standard when storing small thumbnails, the embedded images were easy to locate and carve out of a given thumbnail cache.

For this project, the initial concept was first proven by copying data straight out of a hex editor and pasting it back into a newly created file with a JPG extension. The next step was to automate this process through a simple program. Due to previous experience with the language, Java was chosen for this initial work. Once the preliminary algorithms had been created, and when addition functionality was deemed necessary, the program was then migrated to VC++ for further development. Through this process, huge strides were made toward the creation of a professional-grade tool.

With further development and refinement, this project could evolve into an essential tool for forensic investigation in both the professional and amateur realm. There is a great deal of work left to be done, but a solid base has been set towards this goal.

11. Appendix

11.1 Java Prototype

11.1.1 ThumbScan.java

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.util.Calendar;
import java.text.SimpleDateFormat;

/**
 *    Thumbs.db Scanner <br><br>
 *    Manages the overall operation of the Thumbs.db scanner. All searching is done
 *    by the FindFiles class. All file information is stored in the Info class. <br><br>
 *    The overall program iterates through all subdirectories of a directory in search of
 *    Thumbs.db files. These are then searched for picture data streams. Each discovered
 *    stream is copied into a separate picture file stored in the "Thumbs" directory. All
 *    Information on all Thumbs.db files and pictures found is printed to the screen and
 *    recorded in the ScannerLog.txt file in the same directory as the Scanner.
 *
 *    @author Joe Elinski
 *    @version 1.0
 */
public class ThumbScan{
      private FindFiles find;
      private LinkedList <Info> data;
      private ArrayList <File> hiddenFiles;
      private LinkedList <String> output;
      private final String FILE_NAME = "ScannerLog.txt";

      /**
       *    Sole constructor. Initializes the Scanner, FindFiles, and Info objects.
       *  Starts and manages the scanning process.
       *    @param file The starting directory - taken in as a command line argument.
       */
      public ThumbScan(File search){
```

```
        //instantiate the output list to store all important output
        output = new LinkedList();

        //store the current time to calculate the run time of this scanner
      long startTime = System.currentTimeMillis();

        //set the directory variable
        File dir = search;

        //print starting banner
        printStatus(0, dir.getAbsolutePath());

        //get and store the current date and time
     Calendar cal = Calendar.getInstance();
SimpleDateFormat sdf = new SimpleDateFormat("MM-dd-yyyy HH:mm:ss");
        output.add("Scan started at: " + sdf.format(cal.getTime()));

     //Use the FindFiles class to search and test files
        find = new FindFiles();             //used to do all searching

     System.out.println("\t\tSearching SubDirectories.");
        data = find.search(dir);       //store all found Thumbs files in 'data'



        //print number of files found
        printStatus(2, null);
        printStatus(3, Integer.toString(data.size()));

        //pint out and save list of Thumbs files
        for (int x = 0; x< data.size(); x++){

            //Print Parent of Thumbs file
            printStatus(4, data.get(x).getParent());

            //Print number of found files
            String temp = new String("" + data.get(x).getPictureNumbers().size());
```

```
            printStatus(5, temp);

            //Print Images associated with Thumbs file
            ArrayList pics = data.get(x).getPictureNumbers();
            for (int z = 0; z < pics.size(); z++){
                  printStatus(6, pics.get(z).toString());
            }
      }


      //calculate and print run time
      long endTime = System.currentTimeMillis();
      printStatus(7, Long.toString((endTime-startTime)/1000));


      //write all ouput data to a log File
      save();

      //print end of program
      printStatus(8, null);
}



/**
 *    writes all console data to log file
 */
public void save(){

      //create the printWriter
      PrintWriter out = null;

      //create PrintWriter
      try{
            File file = new File(FILE_NAME);
            out = new PrintWriter(new FileWriter(file), true);
```

```
            for (int x = 0; x<output.size(); x++){
                out.println(output.get(x));
            }

            //close the stream
            out.close();

            //print that the output file was written
            printStatus(1, FILE_NAME);
        }
        catch(IOException ioe){
            ioe.getMessage();
        }
}

/**
 *  Prints the status of the program based on the number and info given
 *  @param where an int that relates to the position in the program
 *  @param what the extra info to be printed
 */
private void printStatus(int where, String what){
        switch(where){
            case 0: System.out.println("\n************************************");
                    System.out.println("Thumbs.db Scanner \n");
                    System.out.println("************************************\n");
                    System.out.println("Starting scan of: " + what);
                    output.add("************************************\n");
                    output.add("Thumbs.db Scanner \n");
                    output.add("************************************\n");
                    output.add("Starting scan of: " + what);
                    break;
            case 1: System.out.println("Writing output file: " + what );
                    System.out.println("************************************");
                    break;
            case 2: System.out.println("\nScan Finished.\n");
                    System.out.println("************************************");
                    output.add("Scan Finished.\n");
```

```
                output.add("************************************");
                break;
        case 3: System.out.println("Thumbs.db Files: " + what );
                output.add("Thumbs.db Files: " + what );
                break;
        case 4: System.out.println("\nLocation: " + what);
                output.add(" ");
                output.add("Location: " + what);
                break;
        case 5: System.out.println("      Total Files: " + what );
                output.add("  Total Files: " + what );
                break;
        case 6: System.out.println("      Associated Picture: " + what);
                output.add("  Associated Picture: " + what);
                break;
        case 7:   System.out.println("\nProcessing took: " + what + " seconds");
                System.out.println("************************************");
                output.add(" ");
                output.add("Processing took: " + what + " seconds");
                output.add("************************************");
                break;
        case 8:
                System.out.println("Done.");
                break;
   }
}

/**
 *    Takes in command line arguments and checks for argument validity and format.
 */
public static void main(String [] args){

      //the desired root path
      File directory = null;

      //default path and signature file, in case they are not provided
    String defaultPath = "C:\\";
```

```
//create a boolean to track whether user has inputed a dir file yet
//this will keep them from attempting to name two default directories, etc.
boolean haveDir = false;


//loop through arguments, pulling out necessary information
//if bad flags are found, then display usage
for (int x = 0; x < args.length; x++){
      if(args[x].equals("-d")){
            if (!haveDir){
                  directory = new File(args[x + 1]);
                  x++;
                  haveDir = true;
            }
      }
      else{
            System.out.println("\n**************************");
            System.out.println("Thumbs.db Scanner");
            System.out.println("**************************\n");
            System.out.println("Usage: ThumbScan [-d <Starting Directory>]\n");
            System.out.println("Default Starting Directory: C:\\");
            System.exit(1);
      }

}

//if no directory was given, then check and use default
if (directory == null){
   directory = new File(defaultPath);

}

//check integrity of starting directory
if (!directory.isDirectory()){
      if (!directory.exists()){
            System.out.println("Error: " + directory + " does not exist.");
```

```
                System.exit( 1 );
            }
            else{
                System.out.println("Error: " + directory + " is not a directory.");
                System.exit( 1 );
            }
        }


        //start the scanner using the determined directory and signature file
        new ThumbScan(directory);
    }

}
```

11.1.2 FindFiles.java

```
import java.io.*;
import java.util.*;
import java.util.zip.*;

/**
 *  Iterates through all subdirectories of a directory and searchs for the presence
 *  of Thumbs.db files. When found, each is searched for the presence of JFIF headers.
 *  Any such JPEG data streams are then copied into individual picture files in a
 *  "Thumbs" directory.
 *
 *  @author Joe Elinski
 *  @version 1.0
 */
public class FindFiles{
    private LinkedList <Info> data;
    private FileInputStream in;
    private File sigFile;
    private int counter;

    /**
     *  The sole constructor. Automates the class.
```

```
 */
public FindFiles(){

      //initialize the linked list of info objects
      data = new  LinkedList<Info>();

      //initialize the counter to keep track of how many pictures have been found
      counter = 1;
}

/**
 *    Recursively iterates through all subdirectories of the given directory "file". All
 *    Thumbs.db files found are read and searched for JFIF headers. When such a header
 *    is located, the proceding stream of data is removed and placed into a picture file.
 *    @param file the starting directory
 *    @return A LinkedList of Info objects containing information on each discovered Thumbs.db file
 */
public LinkedList search(File file){

      //create the storage directory
      new File("Thumbs").mkdir();

      //initialize the proper file variables
      File[] files;    //an array of files
   files = file.listFiles();     //fill files with all files found in the directory

   //create the reader and string buffer for input
   FileInputStream reader;
   StringBuilder sb;



   //loop through 'files' and check each file found. If the file is a
   //directory, then search that directory. Scan each file, and skip
   //files that are over the String Builder limit (to avoid heap overflows)
   for (int i = 0; i < files.length; i++) {

      //define file 'i' found in 'files'
```

```
File f = new File(files[i].getAbsolutePath());

//if f is a directory then search agan
if (f.isDirectory()){

      //let user know that a directory was found
      System.out.print("\r                                                    ");
      System.out.print("\r\t\tPics Found: " + counter);

      //search the discovered directory
      search(f);
 }
//otherwise, check files against signature array
 else if (f.getName().equals("Thumbs.db")){
       try{
            //initialize the reader and string builder
                  reader = new FileInputStream(f);
                  sb = new StringBuilder();

                  //create an array list to store the names of pictures
                  //held within this Thumbs.db file
                  ArrayList <String> picNums = new ArrayList();

                  //read the file line by line until the end is reached
                  int current;
                  while(  (current = reader.read()) != -1 ){
                        //append each line to the string builder
                        sb.append((char)current);
                  }


                  //initialize the test char array and test string
                  // this is used to help search for the string "JFIF"
                  char testChar[] = {'a','b','c','d'};
                  String test;
```

```
//seach the string buffer for JFIF tags. Pull out pictures
//each time this tag is found..
for (int x = 0; x < sb.length() - 3; x++){


    //grab four chars to check for a match of "JFIF"
    testChar[0] = sb.charAt(x);
    testChar[1] = sb.charAt(x+1);
    testChar[2] = sb.charAt(x+2);
    testChar[3] = sb.charAt(x+3);

    //store the chars in one string
    test = new String(testChar);

    //if you find a JFIF tag, start pulling out the associated picture
    if (test.equals("JFIF")){

        //write picture files
        try{

            //create the new file to hold the JPEG stream
            File pic = new File("Thumbs", counter + ".jpeg");

            //increment the "pics found" counter
            counter++;

            //create the output stream
            FileOutputStream out =  new FileOutputStream(pic);

            //create w to start searching from the current byte
            int w = x+1;

            //create booleans to match the given known extra header signatures
            //  start them all at false, since none have been found yet
            boolean eh1 = false;
            boolean eh2 = false;
            boolean eh3 = false;
```

```
boolean eh4 = false;
boolean eh5 = false;
boolean eh6 = false;
boolean eh7 = false;
boolean header = false;


//while true, write data byte by byte to a new pic file
//error check the data just before writing
while (true){

    //write to the file until you reach the end
    if ( w < sb.length()){


    //If you found extra data in the header, then search unil the
    //extra data ends. When it does, write the given byte and
    //continue as normal.
        if (eh1){
            if (sb.codePointAt(w-6) == 0x68){
                if (sb.codePointAt(w-5) == 0x69){
                    if (sb.codePointAt(w-4) == 0x6A){

                        //end of extra header
                        eh1 = false;
                        out.write(sb.codePointAt(w-7));
                        w++;
                        continue;
                    }
                }
            }
            w++;
        }
        if (eh2){
            if (sb.codePointAt((w-7)) == 0x51){
                if (sb.codePointAt(w-6) == 0x61){
                    if (sb.codePointAt(w-5) == 0x07){
```

```
                    if (sb.codePointAt(w-4) == 0x22){

                        //end of extra header2
                        eh2 = false;
                    out.write(sb.codePointAt(w-7));
                        w++;
                        continue;
                    }
                }
            }
        }
        w++;
    }
    if (eh3){
        if (sb.codePointAt((w-7)) == 0x1F){
            if (sb.codePointAt(w-6) == 0x00){
                if (sb.codePointAt(w-5) == 0x00){
                    if (sb.codePointAt(w-4) == 0x01){

                        //end of extra header3
                        eh3 = false;
                        out.write(sb.codePointAt(w-7));
                        w++;
                        continue;
                    }
                }
            }
        }
        w++;
    }
    if (eh4){
        if (sb.codePointAt((w-7)) == 0x0A){
            if (sb.codePointAt(w-6) == 0x0C){
                if (sb.codePointAt(w-5) == 0x0C){
                    if (sb.codePointAt(w-4) == 0x0B){

                        //end of extra header4
```

```
                    eh4 = false;
                    out.write(sb.codePointAt(w-7));
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}
if (eh5){
    if (sb.codePointAt((w-7)) == 0x14){
        if (sb.codePointAt(w-6) == 0x14){
            if (sb.codePointAt(w-5) == 0x14){
                if (sb.codePointAt(w-4) == 0x14){

                    //end of extra header4
                    eh5 = false;
                    out.write(sb.codePointAt(w-7));
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}

if (eh6){
    if (sb.codePointAt((w-7)) == 0xDA){
        if (sb.codePointAt(w-6) == 0xE1){
            if (sb.codePointAt(w-5) == 0xE2){
                if (sb.codePointAt(w-4) == 0xE3){

                    //end of extra header4
                    eh6 = false;
                    out.write(sb.codePointAt(w-7));
```

```
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}
if (eh7){
    if (sb.codePointAt((w-7)) == 0x04){
        if (sb.codePointAt(w-6) == 0x03){
            if (sb.codePointAt(w-5) == 0x04){
                if (sb.codePointAt(w-4) == 0x07){

                    //end of extra header4
                    eh7 = false;
                    out.write(sb.codePointAt(w-7));
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}




//if you haven't found an extra header, then continue
//to search for them. If none are found in this iteration
//then write the current byte
if (!eh1 && !eh2 && !eh3 && !eh4 && !eh5 && !eh6 && !eh7){

    //if you found the end of the picture, break
    if (sb.codePointAt(w-9) == 0xFF){
        if (sb.codePointAt(w-8) == 0xD9){
```

```
            if (sb.codePointAt(w-7) == 0x00){
                if (sb.codePointAt(w-6) == 0x00){
                    picNums.add((counter-1) + ".jpeg");
                    break;
                }
            }
        }
    }

    //if you passed your header and found another, break
    if (header){
        if (sb.codePointAt(w-9) == 0xFF){
            if (sb.codePointAt(w-8) == 0xD8){
                if (sb.codePointAt(w-7) == 0xFF){
                    if (sb.codePointAt(w-6) == 0xE0){
                        picNums.add((counter-1) + ".jpeg");
                        break;
                    }
                }
            }
        }
    }

    //mark once you've passed your header
    if (sb.codePointAt(w-9) == 0xFF){
            if (sb.codePointAt(w-8) == 0xD8){
                if (sb.codePointAt(w-7) == 0xFF){
                    if (sb.codePointAt(w-6) == 0xE0){
                        header = true;
                        w++;
                    }
                }
            }
        }
```

```
//******
//The following if blocks target known header patterns
//if the pattern doesn't match, then data is skipped until it does
//******
if (sb.codePointAt(w-10) == 0x64){
      if (sb.codePointAt(w-9) == 0x65){
            if (sb.codePointAt(w-8) == 0x66){
                  if (sb.codePointAt(w-7) != 0x67){
                        //found the extra header. Mark until it ends.
                        eh1 = true;
                        w++;
                        continue;
                  }
            }
      }
}
if (sb.codePointAt(w-10) == 0x41){
      if (sb.codePointAt(w-9) == 0x06){
            if (sb.codePointAt(w-8) == 0x13){
                  if (sb.codePointAt(w-7) != 0x51){
                        //found the extra header. Mark until it ends.
                        eh2 = true;
                        w++;
                        continue;
                  }
            }
      }
}


if (sb.codePointAt(w-10) == 0xFF){
      if (sb.codePointAt(w-9) == 0xC4){
            if (sb.codePointAt(w-8) == 0x00){
                  if ( (sb.codePointAt(w-7) != 0x1F) &&
                        (sb.codePointAt(w-7) != 0xB5)){
                        //found the extra header. Mark until it ends.);
```

```
                    eh3 = true;
                    w++;
                    continue;
                }
            }
        }
    }


    if (sb.codePointAt(w-10) == 0x06){
        if (sb.codePointAt(w-9) == 0x08){
            if (sb.codePointAt(w-8) == 0x0C){
                if (sb.codePointAt(w-7) != 0x0A){
                    //found the extra header. Mark until it ends.
                    eh4 = true;
                    w++;
                    continue;
                }
            }
        }
    }


    if (sb.codePointAt(w-10) == 0x14){
        if (sb.codePointAt(w-9) == 0x14){
            if (sb.codePointAt(w-8) == 0x14){
                //if ( (sb.codePointAt(w-7) == 0x81) ||
                     (sb.codePointAt(w-7) == 0x31)
                    || (sb.codePointAt(w-7) == 0x13)){
                if (sb.codePointAt(w-7) != 0x14 &&
                    sb.codePointAt(w-7) != 0xff){
                    //found the extra header. Mark until it ends.
                    eh5 = true;
                    w++;
                    continue;
                }
            }
```

```
                }
            }

            if (sb.codePointAt(w-10) == 0xD7){
                if (sb.codePointAt(w-9) == 0xD8){
                    if (sb.codePointAt(w-8) == 0xD9){
                        if (sb.codePointAt(w-7) != 0xDA){
                            //found the extra header. Mark until it ends.
                            eh6 = true;
                            w++;
                            continue;
                        }
                    }
                }
            }

            if (sb.codePointAt(w-10) == 0x01){
                if (sb.codePointAt(w-9) == 0x02){
                    if (sb.codePointAt(w-8) == 0x04){
                        if (sb.codePointAt(w-7) != 0x04){
                            //found the extra header. Mark until it ends.
                            eh7 = true;
                            w++;
                            continue;
                        }
                    }
                }
            }

            //if no problems were found, then write the current byte
            out.write(sb.codePointAt(w-7));
            w++;
            continue;
        }

    }
    else {
```

```
                                        //reached the end of the Thumbs.db file. Break.
                                        break;
                                    }

                                }
                                //close the stream
                                out.close();
                            }
                            catch(IOException ioe){
                                ioe.getMessage();
                            }

                        }
                    }

                    //add the current thumbs.db info into an Info object
                    data.add(new Info(f, picNums));

                    //close the buffered reader
                    reader.close();
                }

                catch(IOException ex) {
                        ex.printStackTrace();
                }
            }
        }
          //return the ArrayList of all Info objects
            return data;
    }

}
```

11.1.3 Info.java

```
import java.io.*;
import java.util.*;
```

```
/**
*    Holds information for a file (name, path, size, modified, etc.)
*
*    @author Joe Elinski
*    @version 1.0
*/
public class Info{

      private File file;
      private String fileName;
      private String filePath;
      private boolean fileHidden;
      private long fileSize;
      private long timestamp;
      private ArrayList picNums;

      /**
       *    Main constructor
       *    @param file The file to be indexed.
       */
       public Info( File file, ArrayList pictureNums){
            this.file = file;
            picNums = pictureNums;

      }

       /**
       *    Secondary constructor. Simply sets signature match to null.
       *    @param file The file to be indexed.
       */
      public Info (File file){
            this.file = file;
      }

      /**
       *    Accessor method.
```

```
     *    @return The file name.
     */
    public String getFileName(){
        return file.getName();
    }

    /**
     *    Accessor method
     *    @return The absolute file path.
     */
    public String getFilePath(){
        return file.getAbsolutePath();
    }

     /**
     *    Accessor method.
     *    @return The file path.
     */
    public String getPath(){
        return file.getPath();
    }

    /**
     *    Accessor method.
     *    @return The file size.
     */
    public long getFileSize(){
        return file.length();
    }

    /**
     *    Accessor method.
     *    @return Whether the file is hidden or not.
     */
    public boolean getFileHidden(){
        return file.isHidden();
    }
```

```
    /**
     *    Accessor method.
     *    @return The time and date that the file was last modified.
     */
    public long getTimestamp(){
        return file.lastModified();
    }

    /**
     *    Accessor method.
     *    @return The parent of the file.
     */
    public String getParent(){
        return file.getParent();
    }

    /**
     *    Accessor method.
     *    @return The parent file of the file.
     */
    public File getParentFile(){
        return file.getParentFile();
    }

    /**
     *    Accessor method.
     *    @return The parent file of the file.
     */
    public ArrayList getPictureNumbers(){
        return picNums;
    }
}
```

11.1.4 Protype Command Line Output

```
C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbScan\source
>java ThumbScan -d "C:\Documents and Settings\Joe Elinski\My Documents\Programm
ing\ThumbScan"

**********************************
Thumbs.db Scanner

**********************************

Starting scan of: C:\Documents and Settings\Joe Elinski\My Documents\Programmin
g\ThumbScan
        Searching SubDirectories.
        Pics Found: 11
Scan Finished.

**********************************
Thumbs.db Files: 2

Location: C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbS
can\docs\resources
    Total Files: 1
    Associated Picture: 1.jpeg

Location: C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbS
can\sample
    Total Files: 9
    Associated Picture: 2.jpeg
    Associated Picture: 3.jpeg
    Associated Picture: 4.jpeg
```

Associated Picture: 5.jpeg

Associated Picture: 6.jpeg

Associated Picture: 7.jpeg

Associated Picture: 8.jpeg

Associated Picture: 9.jpeg

Associated Picture: 10.jpeg

Processing took: 1 seconds

****

Writing output file: ScannerLog.txt

****

Done.

11.1.5 Prototype log output

****

Thumbs.db Scanner

****

Starting scan of: C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbScan

Scan started at: 02-20-2010 15:26:51

Scan Finished.

****

Thumbs.db Files: 2

Location: C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbScan\docs\resources

Total Files: 1

Associated Picture: 1.jpeg

Location: C:\Documents and Settings\Joe Elinski\My Documents\Programming\ThumbScan\sample

Total Files: 9

Associated Picture: 2.jpeg

Associated Picture: 3.jpeg

Associated Picture: 4.jpeg

Associated Picture: 5.jpeg

Associated Picture: 6.jpeg

Associated Picture: 7.jpeg

Associated Picture: 8.jpeg

Associated Picture: 9.jpeg

Associated Picture: 10.jpeg

Processing took: 0 seconds

****

11.2 ThumbScan Code

11.2.1 Form1.h

```
#pragma once

#include "Scanner.h"
#include "stdlib.h"

namespace ThumbScan {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
```

```
    using namespace System::Drawing;
    using namespace std;
}
    /// <summary>
    /// Summary for Form1
    ///
    /// WARNING: If you change the name of this class, you will need to change the
    ///          'Resource File Name' property for the managed resource compiler tool
    ///          associated with all .resx files this class depends on.  Otherwise,
    ///          the designers will not be able to interact properly with localized
    ///          resources associated with this form.
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();

            //get and display current run folder
            runFolderText->Text = Directory::GetCurrentDirectory() + "ThumbScan";

            //populate currentFolder
            currentFolder->Text   = "C:\\";

            //saveFolderText->Text = "C:\\Documents and Settings\\Neal Elinski\\My
Documents\\Programming\\ThumbScan\\sample"
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
            }
```

```
        }
    private: System::Windows::Forms::Button^  startButton;
    private: System::Windows::Forms::Button^  cancelButton;
    private: System::Windows::Forms::TextBox^  currentFolder;
    protected:

    protected:

    private: System::Windows::Forms::FolderBrowserDialog^  folderBrowserDialog1;
    private: System::Windows::Forms::CheckBox^  unAllocatedBox;

    private: System::Windows::Forms::Button^  browseStart;
    private: System::Windows::Forms::TextBox^  startingFolderText;

    private: System::Windows::Forms::TextBox^  saveFolderText;

    private: System::Windows::Forms::Button^  browseSave;
    private: System::Windows::Forms::GroupBox^  groupBox1;
    private: System::Windows::Forms::RadioButton^  localSaveRadio;

    private: System::Windows::Forms::RadioButton^  chooseSaveRadio;
    private: System::Windows::Forms::GroupBox^  groupBox2;
    private: System::Windows::Forms::RadioButton^  chosenSearchRadio;

    private: System::Windows::Forms::RadioButton^  searchLocalRadio;
    private: System::Windows::Forms::TextBox^  runFolderText;
    private: System::Windows::Forms::RadioButton^  quickScanRadio;

    private: System::Windows::Forms::GroupBox^  groupBox3;



    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
```

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {
            this->startButton = (gcnew System::Windows::Forms::Button());
            this->cancelButton = (gcnew System::Windows::Forms::Button());
            this->currentFolder = (gcnew System::Windows::Forms::TextBox());
            this->folderBrowserDialog1 = (gcnew
System::Windows::Forms::FolderBrowserDialog());
            this->unAllocatedBox = (gcnew System::Windows::Forms::CheckBox());
            this->browseStart = (gcnew System::Windows::Forms::Button());
            this->startingFolderText = (gcnew System::Windows::Forms::TextBox());
            this->saveFolderText = (gcnew System::Windows::Forms::TextBox());
            this->browseSave = (gcnew System::Windows::Forms::Button());
            this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
            this->runFolderText = (gcnew System::Windows::Forms::TextBox());
            this->localSaveRadio = (gcnew System::Windows::Forms::RadioButton());
            this->chooseSaveRadio = (gcnew System::Windows::Forms::RadioButton());
            this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
            this->quickScanRadio = (gcnew System::Windows::Forms::RadioButton());
            this->chosenSearchRadio = (gcnew System::Windows::Forms::RadioButton());
            this->searchLocalRadio = (gcnew System::Windows::Forms::RadioButton());
            this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
            this->groupBox1->SuspendLayout();
            this->groupBox2->SuspendLayout();
            this->groupBox3->SuspendLayout();
            this->SuspendLayout();
            //
            // startButton
            //
            this->startButton->Location = System::Drawing::Point(190, 211);
            this->startButton->Name = L"startButton";
            this->startButton->Size = System::Drawing::Size(106, 32);
            this->startButton->TabIndex = 0;
            this->startButton->Text = L"Start";
            this->startButton->UseVisualStyleBackColor = true;
```

```
            this->startButton->Click += gcnew System::EventHandler(this,
&Form1::startButton_Click);
            //
            // cancelButton
            //
            this->cancelButton->Location = System::Drawing::Point(370, 211);
            this->cancelButton->Name = L"cancelButton";
            this->cancelButton->Size = System::Drawing::Size(103, 32);
            this->cancelButton->TabIndex = 1;
            this->cancelButton->Text = L"Cancel";
            this->cancelButton->UseVisualStyleBackColor = true;
            this->cancelButton->Click += gcnew System::EventHandler(this,
&Form1::cancelButton_Click);
            //
            // currentFolder
            //
            this->currentFolder->BackColor = System::Drawing::SystemColors::ControlLight;
            this->currentFolder->Location = System::Drawing::Point(6, 19);
            this->currentFolder->Name = L"currentFolder";
            this->currentFolder->ReadOnly = true;
            this->currentFolder->Size = System::Drawing::Size(594, 20);
            this->currentFolder->TabIndex = 2;
            //
            // folderBrowserDialog1
            //
            this->folderBrowserDialog1->RootFolder =
System::Environment::SpecialFolder::MyComputer;
            //
            // unAllocatedBox
            //
            this->unAllocatedBox->AutoSize = true;
            this->unAllocatedBox->Location = System::Drawing::Point(170, 21);
            this->unAllocatedBox->Name = L"unAllocatedBox";
            this->unAllocatedBox->Size = System::Drawing::Size(120, 17);
            this->unAllocatedBox->TabIndex = 3;
            this->unAllocatedBox->Text = L"Search Unallocated";
            this->unAllocatedBox->UseVisualStyleBackColor = true;
            //
            // browseStart
```

```
            //
            this->browseStart->Location = System::Drawing::Point(215, 104);
            this->browseStart->Name = L"browseStart";
            this->browseStart->Size = System::Drawing::Size(75, 23);
            this->browseStart->TabIndex = 4;
            this->browseStart->Text = L"Browse";
            this->browseStart->UseVisualStyleBackColor = true;
            this->browseStart->Click += gcnew System::EventHandler(this,
&Form1::browseStart_Click);
            //
            // startingFolderText
            //
            this->startingFolderText->BackColor =
System::Drawing::SystemColors::ControlLight;
            this->startingFolderText->Location = System::Drawing::Point(6, 106);
            this->startingFolderText->Name = L"startingFolderText";
            this->startingFolderText->ReadOnly = true;
            this->startingFolderText->Size = System::Drawing::Size(191, 20);
            this->startingFolderText->TabIndex = 5;
            this->startingFolderText->Text = L"C:\\";
            //
            // saveFolderText
            //
            this->saveFolderText->BackColor = System::Drawing::SystemColors::ControlLight;
            this->saveFolderText->Location = System::Drawing::Point(6, 107);
            this->saveFolderText->Name = L"saveFolderText";
            this->saveFolderText->ReadOnly = true;
            this->saveFolderText->Size = System::Drawing::Size(186, 20);
            this->saveFolderText->TabIndex = 9;
            this->saveFolderText->Text = L"C:\\ThumbScan";
            //
            // browseSave
            //
            this->browseSave->Location = System::Drawing::Point(214, 105);
            this->browseSave->Name = L"browseSave";
            this->browseSave->Size = System::Drawing::Size(75, 23);
            this->browseSave->TabIndex = 8;
            this->browseSave->Text = L"Browse";
            this->browseSave->UseVisualStyleBackColor = true;
```

```
            this->browseSave->Click += gcnew System::EventHandler(this,
&Form1::browseSave_Click);
            //
            // groupBox1
            //
            this->groupBox1->Controls->Add(this->runFolderText);
            this->groupBox1->Controls->Add(this->localSaveRadio);
            this->groupBox1->Controls->Add(this->chooseSaveRadio);
            this->groupBox1->Controls->Add(this->saveFolderText);
            this->groupBox1->Controls->Add(this->browseSave);
            this->groupBox1->Location = System::Drawing::Point(341, 11);
            this->groupBox1->Name = L"groupBox1";
            this->groupBox1->Size = System::Drawing::Size(295, 133);
            this->groupBox1->TabIndex = 11;
            this->groupBox1->TabStop = false;
            this->groupBox1->Text = L"Save Folder";
            //
            // runFolderText
            //
            this->runFolderText->BackColor = System::Drawing::SystemColors::ControlLight;
            this->runFolderText->Location = System::Drawing::Point(6, 45);
            this->runFolderText->Name = L"runFolderText";
            this->runFolderText->ReadOnly = true;
            this->runFolderText->Size = System::Drawing::Size(283, 20);
            this->runFolderText->TabIndex = 12;
            //
            // localSaveRadio
            //
            this->localSaveRadio->AutoSize = true;
            this->localSaveRadio->Checked = true;
            this->localSaveRadio->Location = System::Drawing::Point(6, 21);
            this->localSaveRadio->Name = L"localSaveRadio";
            this->localSaveRadio->Size = System::Drawing::Size(116, 17);
            this->localSaveRadio->TabIndex = 11;
            this->localSaveRadio->TabStop = true;
            this->localSaveRadio->Text = L"Save in Run Folder";
            this->localSaveRadio->UseVisualStyleBackColor = true;
            this->localSaveRadio->CheckedChanged += gcnew System::EventHandler(this,
&Form1::localSaveRadio_CheckedChanged);
```

```
            //
            // chooseSaveRadio
            //
            this->chooseSaveRadio->AutoSize = true;
            this->chooseSaveRadio->Location = System::Drawing::Point(6, 81);
            this->chooseSaveRadio->Name = L"chooseSaveRadio";
            this->chooseSaveRadio->Size = System::Drawing::Size(105, 17);
            this->chooseSaveRadio->TabIndex = 10;
            this->chooseSaveRadio->Text = L"Choose Location";
            this->chooseSaveRadio->UseVisualStyleBackColor = true;
            this->chooseSaveRadio->CheckedChanged += gcnew System::EventHandler(this,
&Form1::chooseSaveRadio_CheckedChanged);
            //
            // groupBox2
            //
            this->groupBox2->Controls->Add(this->quickScanRadio);
            this->groupBox2->Controls->Add(this->chosenSearchRadio);
            this->groupBox2->Controls->Add(this->searchLocalRadio);
            this->groupBox2->Controls->Add(this->startingFolderText);
            this->groupBox2->Controls->Add(this->unAllocatedBox);
            this->groupBox2->Controls->Add(this->browseStart);
            this->groupBox2->Location = System::Drawing::Point(30, 12);
            this->groupBox2->Name = L"groupBox2";
            this->groupBox2->Size = System::Drawing::Size(296, 132);
            this->groupBox2->TabIndex = 12;
            this->groupBox2->TabStop = false;
            this->groupBox2->Text = L"Scan Type";
            //
            // quickScanRadio
            //
            this->quickScanRadio->AutoSize = true;
            this->quickScanRadio->Location = System::Drawing::Point(6, 51);
            this->quickScanRadio->Name = L"quickScanRadio";
            this->quickScanRadio->Size = System::Drawing::Size(81, 17);
            this->quickScanRadio->TabIndex = 8;
            this->quickScanRadio->Text = L"Quick Scan";
            this->quickScanRadio->UseVisualStyleBackColor = true;
            this->quickScanRadio->CheckedChanged += gcnew System::EventHandler(this,
&Form1::quickScanRadio_CheckedChanged);
```

```
            //
            // chosenSearchRadio
            //
            this->chosenSearchRadio->AutoSize = true;
            this->chosenSearchRadio->Location = System::Drawing::Point(6, 83);
            this->chosenSearchRadio->Name = L"chosenSearchRadio";
            this->chosenSearchRadio->Size = System::Drawing::Size(142, 17);
            this->chosenSearchRadio->TabIndex = 7;
            this->chosenSearchRadio->Text = L"Start in Chosen Directory";
            this->chosenSearchRadio->UseVisualStyleBackColor = true;
            this->chosenSearchRadio->CheckedChanged += gcnew System::EventHandler(this,
&Form1::chosenSearchRadio_CheckedChanged);
            //
            // searchLocalRadio
            //
            this->searchLocalRadio->AutoSize = true;
            this->searchLocalRadio->Checked = true;
            this->searchLocalRadio->Location = System::Drawing::Point(7, 20);
            this->searchLocalRadio->Name = L"searchLocalRadio";
            this->searchLocalRadio->Size = System::Drawing::Size(146, 17);
            this->searchLocalRadio->TabIndex = 6;
            this->searchLocalRadio->TabStop = true;
            this->searchLocalRadio->Text = L"Search Entire Local Drive";
            this->searchLocalRadio->UseVisualStyleBackColor = true;
            this->searchLocalRadio->CheckedChanged += gcnew System::EventHandler(this,
&Form1::searchLocalRadio_CheckedChanged);
            //
            // groupBox3
            //
            this->groupBox3->Controls->Add(this->currentFolder);
            this->groupBox3->Location = System::Drawing::Point(30, 150);
            this->groupBox3->Name = L"groupBox3";
            this->groupBox3->Size = System::Drawing::Size(606, 55);
            this->groupBox3->TabIndex = 13;
            this->groupBox3->TabStop = false;
            this->groupBox3->Text = L"Current Scanner Location";
            //
            // Form1
            //
```

```
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(664, 253);
            this->Controls->Add(this->groupBox3);
            this->Controls->Add(this->groupBox2);
            this->Controls->Add(this->groupBox1);
            this->Controls->Add(this->cancelButton);
            this->Controls->Add(this->startButton);
            this->Name = L"Form1";
            this->Text = L"ThumbScan v0.5";
            this->groupBox1->ResumeLayout(false);
            this->groupBox1->PerformLayout();
            this->groupBox2->ResumeLayout(false);
            this->groupBox2->PerformLayout();
            this->groupBox3->ResumeLayout(false);
            this->groupBox3->PerformLayout();
            this->ResumeLayout(false);

        }
#pragma endregion

private: System::Void startButton_Click(System::Object^  sender, System::EventArgs^  e){
             startButton->Enabled = false;

             //for testing
            //gcnew Scanner("C:\\Documents and Settings\\Neal Elinski\\My
Documents\\Programming\\ThumbScan\\sample\\Thumbs", "C:\\TS", currentFolder,
unAllocatedBox->Checked);


            if (localSaveRadio->Checked){
                gcnew Scanner(currentFolder->Text, runFolderText->Text, currentFolder,
unAllocatedBox->Checked);
             }
             else{
                gcnew Scanner(currentFolder->Text, saveFolderText->Text, currentFolder,
unAllocatedBox->Checked);
             }
}
```

```
private: System::Void browseStart_Click(System::Object^  sender, System::EventArgs^  e) {
           folderBrowserDialog1->ShowDialog();
           String^ temp = gcnew String(folderBrowserDialog1->SelectedPath);
           startingFolderText->Text = temp;
           currentFolder->Text = temp;
         }
private: System::Void browseSave_Click(System::Object^  sender, System::EventArgs^  e) {
           folderBrowserDialog1->ShowDialog();
           saveFolderText->Text = folderBrowserDialog1->SelectedPath;

}
private: System::Void cancelButton_Click(System::Object^  sender, System::EventArgs^  e)
{
            exit(0);
         }

private: System::Void localSaveRadio_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
            browseSave->Enabled = false;
         }
private: System::Void chooseSaveRadio_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
            browseSave->Enabled = true;
         }
private: System::Void searchLocalRadio_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
            browseStart->Enabled = false;

            currentFolder->Text = "C:\\";
         }
private: System::Void quickScanRadio_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
            browseStart->Enabled = false;

            if (Directory::Exists("C:\\Users")){
                currentFolder->Text = "C:\\Users";
            }
```

```
            else {
                currentFolder->Text = "C:\\Documents and Settings";
            }
            /** attempt to perform an OS fingerprint -- does not include Vista+
            //perform an OS check to determine start folder
            System::OperatingSystem ^osInfo = System::Environment::OSVersion;
            switch(osInfo->Platform)
            {
            case System::PlatformID::Win32Windows:
            {
            currentFolder->Text = "C:\\Documents and Settings";
            }

            case System::PlatformID::Win32NT:
            {
            currentFolder->Text = "C:\\Documents and Settings";
            }
            default:
            {
            currentFolder->Text = "C:\\Users";
                }
            }**/
        }
private: System::Void chosenSearchRadio_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
            browseStart->Enabled = true;
            currentFolder->Text = startingFolderText->Text;
        }
};
```

11.2.2 ThumbScan.cpp

```
// ThumbScan.cpp : main project file.

#include "stdafx.h"
#include "Form1.h"

using namespace System;
using namespace System::Collections;
```

```
using namespace ThumbScan;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
    // Enabling Windows XP visual effects before any controls are created
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    // Create the main window and run it
    Application::Run(gcnew Form1());

    return 0;
}
```

11.2.3 Scanner.h

```
#pragma once
#include <Windows.h>
#include "NTFSDrive.h"
#include "MFTRecord.h"

using namespace System;
using namespace System::IO;
using namespace System::Collections;
using namespace System::Text;
using namespace System::Drawing;
using namespace std;


typedef struct
{
    WORD   wCylinder;
    WORD   wHead;
    WORD   wSector;
    DWORD  dwNumSectors;
    WORD   wType;
    DWORD  dwRelativeSector;
```

```
    DWORD  dwNTRelativeSector;
    DWORD  dwBytesPerSector;

}DRIVEPACKET;

typedef struct
{
    BYTE   chBootInd;
    BYTE   chHead;
    BYTE   chSector;
    BYTE   chCylinder;
    BYTE   chType;
    BYTE   chLastHead;
    BYTE   chLastSector;
    BYTE   chLastCylinder;
    DWORD  dwRelativeSector;
    DWORD  dwNumberSectors;

}PARTITION;

#define PART_TABLE 0
#define BOOT_RECORD 1
#define EXTENDED_PART 2

#define PART_UNKNOWN 0x00      //Unknown.
#define PART_DOS2_FAT 0x01     //12-bit FAT.
#define PART_DOS3_FAT 0x04     //16-bit FAT. Partition smaller than 32MB.
#define PART_EXTENDED 0x05     //Extended MS-DOS Partition.
#define PART_DOS4_FAT 0x06     //16-bit FAT. Partition larger than or equal to 32MB.
#define PART_DOS32 0x0B        //32-bit FAT. Partition up to 2047GB.
#define PART_DOS32X 0x0C    //Same as PART_DOS32(0Bh), but uses Logical Block Address Int
13h extensions.
#define PART_DOSX13 0x0E    //Same as PART_DOS4_FAT(06h), but uses Logical Block Address
Int 13h extensions.
#define PART_DOSX13X 0x0F      //Same as PART_EXTENDED(05h), but uses Logical Block
Address Int 13h extensions.


ref class Scanner
```

```
{
public:
    Scanner(void);
    Scanner(String^, String^, System::Windows::Forms::TextBox^, bool unDel);
    void run();
private:
    String^ startFolder;
    String^ saveFolder;
    System::Windows::Forms::TextBox^ currentFolder;
    bool unDelete;
    bool ua;
    int counter;
    int uaCounter;
    int numFiles;
    ArrayList^ fileInf;
    void collect(DirectoryInfo^);
    void collectDeleted();
    void analyze(FileInfo^);
    void analyzeStructure(FileInfo^);
};
```

11.2.4 Scanner.cpp

```
#include "StdAfx.h"
#include "Scanner.h"
#include <string.h>
#include "stdlib.h"
#include <stdio.h>
#include <winioctl.h>
#include <vcclr.h>

//not used
Scanner::Scanner(void)
{
    startFolder = "C:\\";
    saveFolder = "ThumbScan\\";
    run();
}

//main constructor
```

```
Scanner::Scanner(String^ startF, String^ saveF, System::Windows::Forms::TextBox^ cur,
bool Undel)
{
    startFolder = startF;
    saveFolder = saveF;
    currentFolder = cur;
    unDelete = Undel;
    counter = 1;
    uaCounter = 1;
    ua = false;

    fileInf = gcnew ArrayList();
    fileInf->Add("Starting Scan of " + startF + "\n");
    fileInf->Add("********************************\n");
    if (!(Directory::Exists(saveF))){
        Directory::CreateDirectory(saveF);
    }
    if (!(Directory::Exists(saveF + "\\UA\\"))){
        Directory::CreateDirectory(saveF + "\\UA\\");
    }

    run();
}

//initialize the scanner and call all main functions
void Scanner::run()
{
    //get start time so later we can calculate total runtime
    unsigned int startTime = ::GetTickCount();

    //if unDelete is selected, then search unallocated space on drive
    //do this first, so the data is not overwritten by other recovered images
    if (unDelete){
        collectDeleted();
    }

    //collect folders, analyze data, and pull out images
    collect(gcnew DirectoryInfo(startFolder));
```

```
    //get end time for total runtime calculations
    unsigned int endTime = ::GetTickCount();

    //inform user that scan has finished and show run time
    System::Windows::Forms::MessageBox::Show("DONE!" + "\n Scan took " +
System::Math::Round((((endTime - startTime) * .001)/60), 2) + " minutes");
    //System::Windows::Forms::MessageBox^ mbox;
    //mbox->Show("DONE!" + "\n Scan took " + System::Math::Round((((endTime - startTime)
* .001)/60), 2) + " minutes");


    StreamWriter^ fOut = gcnew StreamWriter(saveFolder + "\\" + "LOG.txt");
    IEnumerator^ fEnum = fileInf->GetEnumerator();
        while ( fEnum->MoveNext() ){
            fOut->WriteLine(fEnum->Current->ToString());
        }
    fOut->Close();




    //start windows explorer to display the discovered images
    System::Diagnostics::Process^ myProcess = gcnew System::Diagnostics::Process();
    myProcess->StartInfo->FileName = "explorer.exe";
    myProcess->StartInfo->Arguments = "/root," + saveFolder;
    myProcess->Start();

    exit(0);

}

//Collect thumbcaches from the given directory downward
void Scanner::collect(DirectoryInfo^ dir)
{
    try{
        //get all directories and files from the current directory
        array<DirectoryInfo^>^directories = dir->GetDirectories();
        array<FileInfo^>^files = dir->GetFiles();
```

```
        // Search each directory found in the current directory
        IEnumerator^ myEnum = directories->GetEnumerator();

        //If directories are found, then search them
        if (directories->Length > 0){
            while ( myEnum->MoveNext() )
            {
                //get the current directory
                DirectoryInfo^ temp = safe_cast<DirectoryInfo^>(myEnum->Current);

                //display the current directory to the user
                currentFolder->Text = temp->FullName;
                currentFolder->Parent->Update(); //bad practice, ignore this

                //search the current directory
                collect(temp);
            }
        }

        //For each file in the current directory, scan for known Thumbnail caches
        IEnumerator^ fileEnum = files->GetEnumerator();
        while ( fileEnum->MoveNext() ){

            //get the current file
            FileInfo^ currentFile = safe_cast<FileInfo^>(fileEnum->Current);

            //if you find a thumbcache, analyze it
            if (currentFile->Name->Equals("Thumbs.db") || currentFile->Name-
>StartsWith("thumbcache")){
                fileInf->Add("Location: " + currentFile->FullName);
                analyze(currentFile);
            }
         }
    }
    catch (System::UnauthorizedAccessException^){
        //known issue for some files (C/:SystemVolumeInformation...). Just skip the error
for now.
    }
    catch (System::IO::IOException^){
```

```
        //known issue for some files that are opened by OS. Skip for now.
    }
}

void Scanner::collectDeleted(){

    //display the current directory to the user
    currentFolder->Text = "$MFT";
    currentFolder->Parent->Update(); //bad practice, ignore this

    int mon; //monitor -- for return values
    DWORD dwBytes;

    //partition struct -- for holding partition variables
    PARTITION *PartitionTbl;

    //buffers
    BYTE szSector[512];
    WORD wDrive=0;

    //MFT sector information
    DWORD dwMainPrevRelSector=0;
    DWORD dwPrevRelSector=0;

    //create a file to read the MFT into memory
    pin_ptr<const wchar_t> dName = PtrToStringChars("\\\\.\\PhysicalDrive0");
    HANDLE hDrive =
CreateFile((LPCWSTR)dName,GENERIC_READ,FILE_SHARE_READ|FILE_SHARE_WRITE,0,OPEN_EXISTING,0
,0);

    //read
    mon = ReadFile(hDrive,szSector,512,&dwBytes,0);

    //zero out sector variables
    dwPrevRelSector=0;
    dwMainPrevRelSector=0;

    //to hold partition table information
    PartitionTbl = (PARTITION*)(szSector+0x1BE);
```

```
    //setup and initialize the CNTFS object
    CNTFSDrive cntfs;
    cntfs.SetDriveHandle(hDrive); //set the drive to scan
    cntfs.SetStartSector(PartitionTbl->dwRelativeSector, 512); //set start sector
    mon = cntfs.Initialize(); //load the MFT into memory

    //file info object for holding file attributes
    CNTFSDrive::ST_FILEINFO stFInfo;

    for(int i=0;(i<0xFFFFFFFF);i++){

        try{

            //read the file details for the MFC, so we can get the name
            mon = cntfs.GetFileDetail(i+30,stFInfo);
            if (mon == ERROR_NO_MORE_FILES){
                break;
            }
            if (mon){
                continue;
            }
        }
        catch (System::AccessViolationException^){
            continue;  //too bad this is an SEH exception and you can't catch it
        }

        if (stFInfo.bDeleted){
            String^ filName = gcnew String(stFInfo.szFilename);

            if (filName->StartsWith("Thumbs") || filName->StartsWith("thumbcache")){

                //the save path -- must pin or gc will relocate it, throwing an error for
the MFC classes
                String^ stemp = saveFolder + "\\UA\\" + uaCounter + filName +
".recovered.db";
                pin_ptr<const wchar_t> wname = PtrToStringChars(stemp);
```

```
            //create a new blank file to hold the restored Thumbs.db file
            HANDLE hNewFile = CreateFile(  (LPCWSTR)wname,
                GENERIC_WRITE,
                0,
                NULL,
                CREATE_ALWAYS,
                stFInfo.dwAttributes,
                0);


            //create the necessary buffers to read from the MFT
            BYTE *pData;
            DWORD dwBytes =0;
            DWORD dwLen;

            //nRet = m_cNTFS.Read_File(nIdxSelLst+30,pData,dwLen);
            cntfs.Read_File(i+30,pData, dwLen);

            //create the file
            mon = WriteFile(hNewFile,pData,dwLen,&dwBytes,NULL);

            //create a FileInfo handle for the file for processing
            FileInfo^ getDet = gcnew FileInfo(stemp);

            //close the handle, so the analyze class can access the file
            CloseHandle(hNewFile);

            //mark unallocated as true
            ua = true;

            fileInf->Add("Location: " + stemp + "\n");

            //analyze the files in the recovered thumbcache
            analyze(getDet);

            uaCounter++; //count the number of files found from the unallocated space
        }
    }
}
```

```
    if(CloseHandle(hDrive) != 0)
        wprintf(L"\nhVolume handle was closed successfully!\n");
    else
    {
        wprintf(L"\nFailed to close hVolume handle!\n");
    }

}

void Scanner::analyze(FileInfo^ file){

    //create an array list to store the names of pictures
    //held within this Thumbs.db file
    //ArrayList <String^>^ picNums = new ArrayList();
    array<unsigned char>^ sb;

    //read in all the byte data of the file
    if (file->Length > 0){
        sb = File::ReadAllBytes(file->FullName);
    }
    else{
        return;
    }

    //initialize the test char array and test string this is used to help search for the
string "JFIF"
    char testChar[4] = {'J','F','I','F'};
    char match[4] = {'J','F','I','F'};


    numFiles = 0;


    //seach the string buffer for JFIF tags. Pull out pictures
    //each time this tag is found...
    for (int x = 0; x < ((sb->Length) - 3); x++){

           //grab four chars to check for a match of "JFIF"
```

```
            testChar[0] = sb[x];
            testChar[1] = sb[x+1];
            testChar[2] = sb[x+2];
            testChar[3] = sb[x+3];

            //if you find a JFIF tag, start pulling out the associated picture
            if (String::Compare(gcnew String(testChar, 0, 4),gcnew String(match, 0, 4)) ==
            0){


                //create the new file to hold the JPEG stream
                String^ pic = counter + ".jpg";

                //increment the "pics found" counter
                if (ua){
                    uaCounter++;
                }
                else{
                    counter++;
                }


                //create the output stream
                //StreamWriter^ out = gcnew StreamWriter(saveFolder + "\\" + pic);
                BinaryWriter^ out;
                if (ua){
                    out = gcnew BinaryWriter( File::Open( saveFolder + "\\UA\\" + uaCounter
+ ".jpg", FileMode::Create ) );
                    fileInf->Add(saveFolder + "\\UA\\" + uaCounter + ".jpg\n");
                }
                else{
                    out = gcnew BinaryWriter( File::Open( saveFolder + "\\" + pic,
FileMode::Create ) );
                    fileInf->Add(saveFolder + "\\" + pic + "\n");
                }
                //ArrayList^ out = gcnew ArrayList();

                //create w to start searching from the current byte
                int w = x+1;
```

```
//create booleans to match the given known extra header signatures
//  start them all at false, since none have been found yet
bool eh1 = false;
bool eh2 = false;
bool eh3 = false;
bool eh4 = false;
bool eh5 = false;
bool eh6 = false;
bool eh7 = false;
bool header = false;

//StringBuilder^ picStream = sb;
array<unsigned char>^ picStream = sb;

//while true, Write data byte by byte to a new pic file
//error check the data just before writing
while (true){

   //Write to the file until you reach the end
   if ( w < picStream->Length){

   //If you found extra data in the header, then search until the
   //extra data ends. When it does, Write the given byte and
   //continue as normal.
      if (eh1){
         if (picStream[w-6] == 0x68){
            if (picStream[w-5] == 0x69){
               if (picStream[w-4] == 0x6A){

                  //end of extra header
                  eh1 = false;
                  out->Write(picStream[w-7]);
                  w++;
                  continue;
               }
            }
         }
         w++;
```

```
}
if (eh2){
    if (picStream[w-7] == 0x51){
        if (picStream[w-6] == 0x61){
            if (picStream[w-5] == 0x07){
                if (picStream[w-4] == 0x22){

                    //end of extra header2
                    eh2 = false;
                    out->Write(picStream[w-7]);
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}
if (eh3){
    if (picStream[w-7] == 0x1F){
        if (picStream[w-6] == 0x00){
            if (picStream[w-5] == 0x00){
                if (picStream[w-4] == 0x01){

                    //end of extra header3
                    eh3 = false;
                    out->Write(picStream[w-7]);
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}
if (eh4){
    if (picStream[w-7] == 0x0A){
        if (picStream[w-6] == 0x0C){
            if (picStream[w-5]== 0x0C){
```

```
                if (picStream[w-4] == 0x0B){

                    //end of extra header4
                    eh4 = false;
                    out->Write(picStream[w-7]);
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}
if (eh5){
    if (picStream[w-7] == 0x14){
        if (picStream[w-6] == 0x14){
            if (picStream[w-5] == 0x14){
                if (picStream[w-4] == 0x14){

                    //end of extra header4
                    eh5 = false;
                    out->Write(picStream[w-7]);
                    w++;
                    continue;
                }
            }
        }
    }
    w++;
}

if (eh6){
    if (picStream[w-7] == 0xDA){
        if (picStream[w-6] == 0xE1){
            if (picStream[w-5] == 0xE2){
                if (picStream[w-4] == 0xE3){

                    //end of extra header4
                    eh6 = false;
```

```
                out->Write(picStream[w-7]);
                w++;
                continue;
            }
        }
    }
    w++;
}
if (eh7){
    if (picStream[w-7] == 0x04){
        if (picStream[w-6] == 0x03){
            if (picStream[w-5] == 0x04){
                if (picStream[w-4] == 0x07){

                //end of extra header4
                eh7 = false;
                out->Write(picStream[w-7]);
                w++;
                continue;
            }
        }
    }
    w++;
}



//if you haven't found an extra header, then continue
//to search for them. If none are found in this iteration
//then Write the current byte
if (!eh1 && !eh2 && !eh3 && !eh4 && !eh5 && !eh6 && !eh7){

    //if you found the end of the picture, break
    if (picStream[w-9] == 0xFF){
        if (picStream[w-8] == 0xD9){
            if (picStream[w-7] == 0x00){
```

```
            if (picStream[w-6] == 0x00){
                //picNums.add((counter-1) + ".jpeg");
                break;
            }
        }
    }
}

//if you passed your header and found another, break
if (header){
    if (picStream[w-9] == 0xFF){
        if (picStream[w-8] == 0xD8){
            if (picStream[w-7] == 0xFF){
                if (picStream[w-6] == 0xE0){
                    //picNums.add((counter-1) + ".jpeg");
                    break;
                }
            }
        }
    }
}

//mark once you've passed your header
if (picStream[w-9] == 0xFF){
        if (picStream[w-8] == 0xD8){
            if (picStream[w-7] == 0xFF){
                if (picStream[w-6] == 0xE0){
                    header = true;
                }
            }
        }
    }


//******
//The following if blocks target known header patterns
//if the pattern doesn't match, then data is skipped until it
does
//******
```

```
if (picStream[w-10] == 0x64){
    if (picStream[w-9] == 0x65){
        if (picStream[w-8] == 0x66){
            if (picStream[w-7] != 0x67){
                //found the extra header. Mark until it ends.
                eh1 = true;
                w++;
                continue;
            }
        }
    }
}
if (picStream[w-10] == 0x41){
    if (picStream[w-9] == 0x06){
        if (picStream[w-8] == 0x13){
            if (picStream[w-7] != 0x51){
                //found the extra header. Mark until it ends.
                eh2 = true;
                w++;
                continue;
            }
        }
    }
}


if (picStream[w-10] == 0xFF){
    if (picStream[w-9] == 0xC4){
        if (picStream[w-8] == 0x00){
            if ( (picStream[w-7] != 0x1F) && (picStream[w-7] !=
                0xB5)){
                //found the extra header. Mark until it ends.);
                eh3 = true;
                w++;
                continue;
            }
        }
    }
}
```

```
if (picStream[w-10] == 0x06){
    if (picStream[w-9] == 0x08){
        if (picStream[w-8] == 0x0C){
            if (picStream[w-7] != 0x0A){
                //found the extra header. Mark until it ends.
                eh4 = true;
                w++;
                continue;
            }
        }
    }
}


if (picStream[w-10] == 0x14){
    if (picStream[w-9] == 0x14){
        if (picStream[w-8] == 0x14){
            if (picStream[w-7] != 0x14 && picStream[w-7] != 0xff){
                //found the extra header. Mark until it ends.
                eh5 = true;
                w++;
                continue;
            }
        }
    }
}

if (picStream[w-10] == 0xD7){
    if (picStream[w-9] == 0xD8){
        if (picStream[w-8] == 0xD9){
            if (picStream[w-7] != 0xDA){
                //found the extra header. Mark until it ends.
                eh6 = true;
                w++;
                continue;
            }
        }
```

```
                        }
                    }

                    if (picStream[w-10] == 0x01){
                        if (picStream[w-9] == 0x02){
                            if (picStream[w-8] == 0x04){
                                if (picStream[w-7] != 0x04){
                                    //found the extra header. Mark until it ends.
                                    eh7 = true;
                                    w++;
                                    continue;
                                }
                            }
                        }
                    }

                    //if no problems were found, then Write the current byte
                    out->Write(picStream[w-7]);
                    w++;
                    continue;
                }

            }
            else {
                //reached the end of the Thumbs.db file. Break.
                break;
            }

        }

        out->Close();
        numFiles++;
        }
    }
    fileInf->Add("Total Files: " + numFiles + "\n");
    fileInf->Add("\n");
    ua = false;
}
```

```
//currently unfinished class -- writted to extract file data from the thumbcache -- name,
mod date, creation date, etc.
void Scanner::analyzeStructure(FileInfo^ file){

    array<unsigned char>^ sb = File::ReadAllBytes(file->FullName);

    //fileReader->Close();

    //initialize the test char array and test string this is used to help search for the
string "JFIF"
    char testChar[4] = {'J','F','I','F'};
    char match[4] = {'J','F','I','F'};

//*** code for analyzing structure. Get to this later
        //Analyze header block

    //check that thumbcache magic number is correct
    array<unsigned char>^ dbHeader = {0xd0,0xcf,0x11,0xe0,0xa1,0xb1,0x1a,0xe1};
    int headerCount = 0;
    for (int i = 0; i < dbHeader->Length; ++i) {
        for (int j = 0; j < dbHeader->Length; ++j) {
            if (dbHeader[i] == sb[j]) {
                headerCount++;
            }
        }
    }
    if (headerCount != 8){
        return;
    }


    //pull out key header data...

        //The number of sectors in a fat chain
        array<unsigned char>^bigFatBlocks = {sb[0x2c],sb[0x2d],sb[0x2e],sb[0x2f]};
        int numBigFat = System::BitConverter::ToInt32(bigFatBlocks,0);

        //The first sector of the directory chain (root)
        array<unsigned char>^rootEntry = {sb[0x30],sb[0x31],sb[0x32],sb[0x33]};
```

```
int rootLoc = System::BitConverter::ToInt32(rootEntry,0);

//The first sector of the mini-fat chain
array<unsigned char>^smallFatLoc = {sb[0x3c],sb[0x3d],sb[0x3e],sb[0x3f]};
int firstSmallFat = System::BitConverter::ToInt32(smallFatLoc,0);

//The number of sectors of the mini-fat chain
array<unsigned char>^smallFatBlocks = {sb[0x40],sb[0x41],sb[0x42],sb[0x43]};
int numSmallFat = System::BitConverter::ToInt32(smallFatBlocks,0);

//create the special fat table markers
array <unsigned char>^ endOfChain = {0xfe, 0xff, 0xff, 0xff}; //end of chain
array <unsigned char>^ fatSector = {0xfd, 0xff, 0xff, 0xff}; //a section of the
fat sector
array <unsigned char>^ freeSpace = {0xff, 0xff, 0xff, 0xff};//unallocated space

 int eoc = System::BitConverter::ToInt32(endOfChain, 0);
 int fsec = System::BitConverter::ToInt32(fatSector,0);
 int fspa = System::BitConverter::ToInt32(freeSpace,0);

 delete endOfChain;
 delete fatSector;
 delete freeSpace;

//Catalog any FATblocks
//...Loop through the starting index of the fatblock directory, collecting the
location
//...of each fat block. Typically, in a thumbs.db file, this will only contain one
item
//...of all 0's, since there is only 1 fat block at sector 0. It may be a
different value
 //...in larger thumbnail caches
ArrayList^ FATblocks = gcnew ArrayList();
for (int i = 0; i<numBigFat; i++){
    int offset = 0x4c + i*4;

    array <unsigned char>^ tempFBN = {sb[offset], sb[offset+1], sb[offset+2],
    sb[offset+3]};
    FATblocks->Add(System::BitConverter::ToInt32(tempFBN, 0));
```

```
    delete tempFBN;
}

//calculate position of mini fat blocks
//...start at the location of the first mini fat block (given in the header)
//...Loop from there, adding each new discovered chain item until an eoc is found
int i = firstSmallFat;
ArrayList^ MiniFATblocks = gcnew ArrayList();
while (i != eoc){
    MiniFATblocks->Add(i);

    int offset = 0; //instantiate the offset int
    //calculate next miniFATblock to add
    int iFat = i/0x80; //fat block number to search
    int iSect = i%0x80; //sector to search in fat block
    if ((int)FATblocks[0] != 0){
        offset = (((int)FATblocks[iFat]) * 0x200) + 0x200 + (iSect*4);
    }
    else{
        offset = 0x200 + iSect*4;
    }
    array <unsigned char>^ tempFB= {sb[offset], sb[offset+1], sb[offset+2],
    sb[offset+3]};
    i = System::BitConverter::ToInt32(tempFB,0);
}

//move to the root directory entries so we can get the location of the minitfat
    streams
int currentBlock = rootLoc;

//locate start of minifat streams
int offset = 0x200 + currentBlock * 0x200;
array<unsigned char>^ firstMiniFATstreamBlock = {sb[offset + 0x74], sb[offset +
    0x75], sb[offset + 0x76], sb[offset + 0x77]};
int fmfsb = System::BitConverter::ToInt32(firstMiniFATstreamBlock,0);

//calculate position of minifat streams(data)
i = fmfsb;
ArrayList^ MiniFATstreamBlocks = gcnew ArrayList();
```

```
while (i != eoc){
    MiniFATstreamBlocks->Add(i);

    //calculate next miniFATblock to add
    int iFat = i/0x80; //fat block number to search
    int iSect = i%0x80; //sector to search in fat block
    if ((int)FATblocks[0] != 0){
        offset = (((int)FATblocks[iFat]) * 0x200) + 0x200 + (iSect*4);
    }
    else{
        offset = 0x200 + iSect*4;
    }
    array <unsigned char>^ tempFB= {sb[offset], sb[offset+1], sb[offset+2],
    sb[offset+3]};
    i = System::BitConverter::ToInt32(tempFB,0);
}

int SID = 0;
//*******************************
//for counting purposes
int numPic = 0;
    //Search the FAT sectors for important information
    while (currentBlock != eoc){
        int offset = 0x200 + currentBlock * 0x200;
        for (int i = offset; i < offset+0x200; i += 0x80){



            //get the size of the name
            array <unsigned char>^ encNameSize = {sb[i+0x40], sb[i+0x41], 0x00,
            0x00};
            int nameSize = System::BitConverter::ToInt32(encNameSize,0);

            //get the name of the stream
            array <unsigned char>^ encName = gcnew array<unsigned char>(32);
            for (int x=0; x<31; x++){
                if (encName[x] = 0x00){
                    if (encName[x+1] = 0x00){
                        break;
```

```
        }
    }

    encName[x] = sb[i + x];
}
System::Text::UnicodeEncoding^ encUni = gcnew
    System::Text::UnicodeEncoding();
String^ name1 = encUni->GetString(encName);
System::Text::ASCIIEncoding^ encAsc = gcnew
    System::Text::ASCIIEncoding();
array <unsigned char>^ name2 = encAsc->GetBytes(name1);
String^ name = encAsc->GetString(name2);


//::WideCharToMultiByte(CP_ACP, 0, name2, -1, name, 1024, NULL, NULL);


//get the type of stream (1=storage|2=stream|5=root)
array <unsigned char>^ encBType = {sb[i+0x42], 0x00, 0x00, 0x00};
int bType = System::BitConverter::ToInt32(encBType,0);

//get timestamp 1 seconds
array <unsigned char>^ encTS1Sec = {sb[i+0x64], sb[i+0x65], sb[i+0x66],
    sb[i+0x67]};
int ts1Sec = System::BitConverter::ToInt32(encTS1Sec,0);

//get timestamp 1 days
array <unsigned char>^ encTS1Day = {sb[i+0x68], sb[i+0x69], sb[i+0x6a],
    sb[i+0x6b]};
int ts1Day = System::BitConverter::ToInt32(encTS1Day,0);

//get timestamp 2 seconds
array <unsigned char>^ encTS2Sec = {sb[i+0x6c], sb[i+0x6d], sb[i+0x6e],
    sb[i+0x6f]};
int ts2Sec = System::BitConverter::ToInt32(encTS2Sec,0);

//get timestamp 2 days
array <unsigned char>^ encTS2Day = {sb[i+0x70], sb[i+0x71], sb[i+0x72],
    sb[i+0x73]};
```

```
int ts2Day = System::BitConverter::ToInt32(encTS2Day,0);

//get starting block
array <unsigned char>^ encStartBlock = {sb[i+0x74], sb[i+0x75],
    sb[i+0x76], sb[i+0x77]};
int startBlock = System::BitConverter::ToInt32(encStartBlock,0);

//get element size
array <unsigned char>^ encElSize = {sb[i+0x78], sb[i+0x79], sb[i+0x7a],
    sb[i+0x7b]};
int elSize = System::BitConverter::ToInt32(encElSize,0);

//if you found a stream
StringBuilder^ imageData = gcnew StringBuilder(); //holds image stream
int bytesToWrite = elSize; //current number of bytes to store

//type 2 = stream
if (bType == 2){

    //if the file is greater than the miniSectorCutoff, then it is in
    the FAT
    if (elSize >= 4096){
        int currentStreamBlock = startBlock; // starting position
        while (currentStreamBlock != -2){

            //calcuate the current offset to start from
            int sOffset = 0x200 + currentStreamBlock * 0x200;

            //if you have more than 512 bytes, then keep writing in chunks
            of 512
            if (bytesToWrite >= 512){
                for (int y = 0; y < 512; y++){
                    imageData->Append((unsigned char)sb[sOffset + y]);
                }
            }

            //otherwise, if less than 512, just write what's left
            else{
                for (int y = 0; y < bytesToWrite; y++){
```

```
                imageData->Append((unsigned char)sb[sOffset + y]);
            }
        }

        //decrement the number of bytes left by blocks of 512
        bytesToWrite -= 512;

        //calculate next location of the stream
        int iFat = currentStreamBlock/0x80; //fat block number to
        search
        int iSect = currentStreamBlock%0x80; //sector to search in fat
        block
        int offset = ((int)FATblocks[iFat]) * 0x200 + 0x200 + iSect*4;
        array <unsigned char>^ tempFB= {sb[offset], sb[offset+1],
        sb[offset+2], sb[offset+3]};
        currentStreamBlock = System::BitConverter::ToInt32(tempFB,0);
    }
} //end stream of 4096 bytes

//if the file is too small for the FAT, then it is in the minifat
else{
    int currentStreamMiniBlock = startBlock;
    while (currentStreamMiniBlock != -2 ){

        //calc offset of miniblock to copy
        //----------------------------
        //which sector of the miniFat?
        int nSec = currentStreamMiniBlock/8;
        //where is it located?
        int blockLoc = (int)MiniFATstreamBlocks[nSec];
        //what is the offset?
        int ioffset = (currentStreamMiniBlock % 8) * 64;
        int sOffset = 0x200 + blockLoc*0x200 + ioffset;

        if (bytesToWrite >= 64){
            for (int y = 0; y < 64; y++){
                imageData->Append((unsigned char)sb[sOffset + y]);
            }
        }
```

```
                else {
                    for (int y = 0; y < bytesToWrite; y++){
                        imageData->Append((unsigned char)sb[sOffset + y]);
                    }
                }

                bytesToWrite -= 64;

                //calculate next location of the stream
                int iFat = currentStreamMiniBlock/0x80; //fat block number to
                search
                int iSect = currentStreamMiniBlock%0x80; //sector to search in
                fat block
                offset = (((int)MiniFATblocks[iFat]) * 512) + 0x200 +
                 (iSect*4);
                array <unsigned char>^ tempFB= {sb[offset], sb[offset+1],
                    sb[offset+2], sb[offset+3]};
                currentStreamMiniBlock =
                System::BitConverter::ToInt32(tempFB,0);
            }
        } //end stream <4096


        BinaryWriter^ out = gcnew BinaryWriter( File::Open( numPic + ".jpg",
        FileMode::Create ) );
        numPic++;
        for (int w = 0; w < imageData->Length; w++){
            out->Write(imageData[w]);
        }
        out->Close();
    } //end type 2


} // end catalog processing

SID++;//increment Sector ID

//calculate next location of the stream
int iFat = currentBlock/0x80; //fat block number to search
```

```
            int iSect = currentBlock%0x80; //sector to search in fat block
            if ((int)FATblocks[0] != 0){
                offset = (((int)FATblocks[iFat]) * 0x200) + 0x200 + (iSect*4);
            }
            else{
                offset = 0x200 + iSect*4;
            }
            array <unsigned char>^ tempFB= {sb[offset], sb[offset+1], sb[offset+2],
                sb[offset+3]};
            currentBlock = System::BitConverter::ToInt32(tempFB,0);
        }
}
```

11.2.5 NTFSDrive.h

```
// NTFSDrive.h: interface for the CNTFSDrive class.
//
//////////////////////////////////////////////////////////////////////

#include <windows.h>

#if !defined(AFX_NTFSDRIVE_H__078B2392_2978_4C23_97FD_166C4B234BF3__INCLUDED_)
#define AFX_NTFSDRIVE_H__078B2392_2978_4C23_97FD_166C4B234BF3__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#pragma pack(push, curAlignment)

#pragma pack(1)
//////////////////////////  boot sector info ///////////////////////////////////////
struct NTFS_PART_BOOT_SEC
{
    char      chJumpInstruction[3];
    char      chOemID[4];
    char      chDummy[4];
```

```
    struct NTFS_BPB
    {
        WORD       wBytesPerSec;
        BYTE       uchSecPerClust;
        WORD       wReservedSec;
        BYTE       uchReserved[3];
        WORD       wUnused1;
        BYTE       uchMediaDescriptor;
        WORD       wUnused2;
        WORD       wSecPerTrack;
        WORD       wNumberOfHeads;
        DWORD      dwHiddenSec;
        DWORD      dwUnused3;
        DWORD      dwUnused4;
        LONGLONG   n64TotalSec;
        LONGLONG   n64MFTLogicalClustNum;
        LONGLONG   n64MFTMirrLogicalClustNum;
        int        nClustPerMFTRecord;
        int        nClustPerIndexRecord;
        LONGLONG   n64VolumeSerialNum;
        DWORD      dwChecksum;
    } bpb;

    char       chBootstrapCode[426];
    WORD       wSecMark;
};
//////////////////////////////////////////////////////////////////////////////
#pragma pack(pop, curAlignment)


class CNTFSDrive
{
protected:
///////////////// physical drive info//////////
    HANDLE m_hDrive;
    DWORD m_dwStartSector;
    bool m_bInitialized;
```

```
    DWORD m_dwBytesPerCluster;
    DWORD m_dwBytesPerSector;

    int LoadMFT(LONGLONG nStartCluster);

////////////////////// the MFT info //////////////////
    BYTE *m_puchMFT;  /// the var to hold the loaded entire MFT
    DWORD m_dwMFTLen; // size of MFT

    BYTE *m_puchMFTRecord; // 1K, or the cluster size, whichever is larger
    DWORD m_dwMFTRecordSz; // MFT record size

public:
    struct ST_FILEINFO // this struct is to retrieve the file detail from this class
    {
        char szFilename[_MAX_PATH]; // file name
        LONGLONG   n64Create;    // Creation time
        LONGLONG   n64Modify;    // Last Modify time
        LONGLONG   n64Modfil;    // Last modify of record
        LONGLONG   n64Access;    // Last Access time
        DWORD      dwAttributes; // file attribute
        LONGLONG   n64Size;      // no of cluseters used
        bool       bDeleted;     // if true then its deleted file
    };

    int GetFileDetail(DWORD nFileSeq, ST_FILEINFO &stFileInfo);
    int Read_File(DWORD nFileSeq, BYTE *&puchFileData, DWORD &dwFileDataLen);

    void SetDriveHandle(HANDLE hDrive);
    void SetStartSector(DWORD dwStartSector, DWORD dwBytesPerSector);

    int Initialize();
    CNTFSDrive();
    virtual ~CNTFSDrive();

};

#endif // !defined(AFX_NTFSDRIVE_H__078B2392_2978_4C23_97FD_166C4B234BF3__INCLUDED_)
```

11.2.6 NTFSDrive.cpp

```
// NTFSDrive.cpp: implementation of the CNTFSDrive class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "NTFSDrive.h"
#include "MFTRecord.h"


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CNTFSDrive::CNTFSDrive()
{
    m_bInitialized = false;

    m_hDrive = 0;
    m_dwBytesPerCluster = 0;
    m_dwBytesPerSector = 0;

    m_puchMFTRecord = 0;
    m_dwMFTRecordSz = 0;

    m_puchMFT = 0;
    m_dwMFTLen = 0;

    m_dwStartSector = 0;
}

CNTFSDrive::~CNTFSDrive()
{
    if(m_puchMFT)
        delete m_puchMFT;
    m_puchMFT = 0;
    m_dwMFTLen = 0;
}
```

```
void CNTFSDrive::SetDriveHandle(HANDLE hDrive)
{
    m_hDrive = hDrive;
    m_bInitialized = false;
}

// this is necessary to start reading a logical drive
void CNTFSDrive::SetStartSector(DWORD dwStartSector, DWORD dwBytesPerSector)
{
    m_dwStartSector = dwStartSector;
    m_dwBytesPerSector = dwBytesPerSector;
}

// initialize will read the MFT record
///   and passes to the LoadMFT to load the entire MFT in to the memory
int CNTFSDrive::Initialize()
{
    NTFS_PART_BOOT_SEC ntfsBS;
    DWORD dwBytes;
    LARGE_INTEGER n84StartPos;

    n84StartPos.QuadPart = (LONGLONG)m_dwBytesPerSector*m_dwStartSector;

    // point the starting NTFS volume sector in the physical drive
    DWORD dwCur =
SetFilePointer(m_hDrive,n84StartPos.LowPart,&n84StartPos.HighPart,FILE_BEGIN);

    // Read the boot sector for the MFT infomation
    int nRet = ReadFile(m_hDrive,&ntfsBS,sizeof(NTFS_PART_BOOT_SEC),&dwBytes,NULL);
    if(!nRet)
        return GetLastError();

    if(memcmp(ntfsBS.chOemID,"NTFS",4)) // check whether it is realy ntfs
        return ERROR_INVALID_DRIVE;

    /// Cluster is the logical entity
    ///  which is made up of several sectors (a physical entity)
    m_dwBytesPerCluster = ntfsBS.bpb.uchSecPerClust * ntfsBS.bpb.wBytesPerSec;
```

```
    if(m_puchMFTRecord)
        delete m_puchMFTRecord;

    m_dwMFTRecordSz = 0x01<<((-1)*((char)ntfsBS.bpb.nClustPerMFTRecord));
    m_puchMFTRecord = new BYTE[m_dwMFTRecordSz];

    m_bInitialized = true;

    // MFTRecord of MFT is available in the MFTRecord variable
    //   load the entire MFT using it
    nRet = LoadMFT(ntfsBS.bpb.n64MFTLogicalClustNum);
    if(nRet)
    {
        m_bInitialized = false;
        return nRet;
    }
    return ERROR_SUCCESS;
}

//// nStartCluster is the MFT table starting cluster
///    the first entry of record in MFT table will always have the MFT record of itself
int CNTFSDrive::LoadMFT(LONGLONG nStartCluster)
{
    DWORD dwBytes;
    int nRet;
    LARGE_INTEGER n64Pos;

    if(!m_bInitialized)
        return ERROR_INVALID_ACCESS;

    CMFTRecord cMFTRec;

    wchar_t uszMFTName[10];
    mbstowcs(uszMFTName,"$MFT",10);

    // NTFS starting point
    n64Pos.QuadPart = (LONGLONG)m_dwBytesPerSector*m_dwStartSector;
    // MFT starting point
```

```
    n64Pos.QuadPart += (LONGLONG)nStartCluster*m_dwBytesPerCluster;


    //  set the pointer to the MFT start
    nRet = SetFilePointer(m_hDrive,n64Pos.LowPart,&n64Pos.HighPart,FILE_BEGIN);
    if(nRet == 0xFFFFFFFF)
        return GetLastError();

    /// reading the first record in the NTFS table.
    //   the first record in the NTFS is always MFT record
    nRet = ReadFile(m_hDrive,m_puchMFTRecord,m_dwMFTRecordSz,&dwBytes,NULL);
    if(!nRet)
        return GetLastError();

    // now extract the MFT record just like the other MFT table records
    cMFTRec.SetDriveHandle(m_hDrive);
    cMFTRec.SetRecordInfo((LONGLONG)m_dwStartSector*m_dwBytesPerSector,
m_dwMFTRecordSz,m_dwBytesPerCluster);
    nRet = cMFTRec.ExtractFile(m_puchMFTRecord,dwBytes);
    if(nRet)
        return nRet;

    if(memcmp(cMFTRec.m_attrFilename.wFilename,uszMFTName,8))
        return ERROR_BAD_DEVICE; // no MFT file available

    if(m_puchMFT)
        delete m_puchMFT;
    m_puchMFT = 0;
    m_dwMFTLen = 0;

    // this data(m_puchFileData) is special since it is the data of entire MFT file
    m_puchMFT = new BYTE[cMFTRec.m_dwFileDataSz];
    m_dwMFTLen = cMFTRec.m_dwFileDataSz;

    // store this file to read other files
    memcpy(m_puchMFT, cMFTRec.m_puchFileData, m_dwMFTLen);

    return ERROR_SUCCESS;
}
```

```
/// this function if suceeded it will allocate the buufer and passed to the caller
//    the caller's responsibility to free it
int CNTFSDrive::Read_File(DWORD nFileSeq, BYTE *&puchFileData, DWORD &dwFileDataLen)
{
    int nRet;

    if(!m_bInitialized)
        return ERROR_INVALID_ACCESS;

    CMFTRecord cFile;

    // point the record of the file in the MFT table
    memcpy(m_puchMFTRecord,&m_puchMFT[nFileSeq*m_dwMFTRecordSz],m_dwMFTRecordSz);

    // Then extract that file from the drive
    cFile.SetDriveHandle(m_hDrive);
    cFile.SetRecordInfo((LONGLONG)m_dwStartSector*m_dwBytesPerSector,
m_dwMFTRecordSz,m_dwBytesPerCluster);
    nRet = cFile.ExtractFile(m_puchMFTRecord,m_dwMFTRecordSz);
    if(nRet)
        return nRet;


    puchFileData = new BYTE[cFile.m_dwFileDataSz];
    dwFileDataLen = cFile.m_dwFileDataSz;

    // pass the file data, It should be deallocated by the caller
    memcpy(puchFileData,cFile.m_puchFileData,dwFileDataLen);

    return ERROR_SUCCESS;
}


int CNTFSDrive::GetFileDetail(DWORD nFileSeq, ST_FILEINFO &stFileInfo)
{
    int nRet;
```

```
    if(!m_bInitialized)
        return ERROR_INVALID_ACCESS;

    if((nFileSeq*m_dwMFTRecordSz+m_dwMFTRecordSz) >= m_dwMFTLen)
        return ERROR_NO_MORE_FILES;

    CMFTRecord cFile;
    // point the record of the file in the MFT table
    if(::IsBadReadPtr(&m_puchMFT[nFileSeq*m_dwMFTRecordSz], m_dwMFTRecordSz))
    {
        return ERROR_INVALID_ACCESS; // In the debugger this address displays as so don't
try to memcpy
    }
    memcpy(m_puchMFTRecord,&m_puchMFT[nFileSeq*m_dwMFTRecordSz],m_dwMFTRecordSz);

    // read the only file detail not the file data
    cFile.SetDriveHandle(m_hDrive);
    cFile.SetRecordInfo((LONGLONG)m_dwStartSector*m_dwBytesPerSector,
m_dwMFTRecordSz,m_dwBytesPerCluster);
    nRet = cFile.ExtractFile(m_puchMFTRecord,m_dwMFTRecordSz,true);
    if(nRet)
        return nRet;

    // set the struct and pass the struct of file detail
    memset(&stFileInfo,0,sizeof(ST_FILEINFO));
    for (int i = 0 ; i < _MAX_PATH ; i++ ) stFileInfo.szFilename[i] =
cFile.m_attrFilename.wFilename[i];

    stFileInfo.dwAttributes = cFile.m_attrFilename.dwFlags;

    stFileInfo.n64Create = cFile.m_attrStandard.n64Create;
    stFileInfo.n64Modify = cFile.m_attrStandard.n64Modify;
    stFileInfo.n64Access = cFile.m_attrStandard.n64Access;
    stFileInfo.n64Modfil = cFile.m_attrStandard.n64Modfil;

    stFileInfo.n64Size    = cFile.m_attrFilename.n64Allocated;
    stFileInfo.n64Size   /= m_dwBytesPerCluster;
    stFileInfo.n64Size    = (!stFileInfo.n64Size)?1:stFileInfo.n64Size;
```

```
    stFileInfo.bDeleted = !cFile.m_bInUse;

    return ERROR_SUCCESS;
}
```

11.2.7 MFTRecord.h

```
// MFTRecord.h: interface for the CMFTRecord class.
//
//////////////////////////////////////////////////////////////////////

#include <windows.h>

#if !defined(AFX_MFTRECORD_H__04A1B8DF_0EB0_4B72_8587_2703342C5675__INCLUDED_)
#define AFX_MFTRECORD_H__04A1B8DF_0EB0_4B72_8587_2703342C5675__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#pragma pack(push, curAlignment)

#pragma pack(1)

//////////////////////////MFT record header and attribute header ///////////////////////
struct NTFS_MFT_FILE
{
    char       szSignature[4];      // Signature "FILE"
    WORD       wFixupOffset;     // offset to fixup pattern
    WORD       wFixupSize;          // Size of fixup-list +1
    LONGLONG   n64LogSeqNumber; // log file seq number
    WORD       wSequence;        // sequence nr in MFT
    WORD       wHardLinks;          // Hard-link count
    WORD       wAttribOffset;       // Offset to seq of Attributes
    WORD       wFlags;              // 0x01 = NonRes; 0x02 = Dir
    DWORD      dwRecLength;      // Real size of the record
    DWORD      dwAllLength;      // Allocated size of the record
    LONGLONG   n64BaseMftRec;       // ptr to base MFT rec or 0
    WORD       wNextAttrID;      // Minimum Identificator +1
    WORD       wFixupPattern;       // Current fixup pattern
```

```
    DWORD      dwMFTRecNumber;      // Number of this MFT Record
                                    // followed by resident and
                                    // part of non-res attributes
};

typedef struct    // if resident then + RESIDENT
{                 //  else + NONRESIDENT
    DWORD  dwType;
    DWORD  dwFullLength;
    BYTE   uchNonResFlag;
    BYTE   uchNameLength;
    WORD   wNameOffset;
    WORD   wFlags;
    WORD   wID;

    union ATTR
    {
        struct RESIDENT
        {
            DWORD  dwLength;
            WORD   wAttrOffset;
            BYTE   uchIndexedTag;
            BYTE   uchPadding;
        } Resident;

        struct NONRESIDENT
        {
            LONGLONG   n64StartVCN;
            LONGLONG   n64EndVCN;
            WORD       wDatarunOffset;
            WORD       wCompressionSize; // compression unit size
            BYTE       uchPadding[4];
            LONGLONG   n64AllocSize;
            LONGLONG   n64RealSize;
            LONGLONG   n64StreamSize;
            // data runs...
        }NonResident;
    }Attr;
} NTFS_ATTRIBUTE;
```

```
//////////////////////////////////////////////////////////////////////////////////////
/

//////////////////////// Attributes
//////////////////////////////////////////////////
typedef struct
{
    LONGLONG   n64Create;    // Creation time
    LONGLONG   n64Modify;    // Last Modify time
    LONGLONG   n64Modfil;    // Last modify of record
    LONGLONG   n64Access;    // Last Access time
    DWORD      dwFATAttributes;// As FAT + 0x800 = compressed
    DWORD      dwReserved1;  // unknown

} ATTR_STANDARD;

typedef struct
{
    LONGLONG   dwMftParentDir;            // Seq-nr parent-dir MFT entry
    LONGLONG   n64Create;                  // Creation time
    LONGLONG   n64Modify;                  // Last Modify time
    LONGLONG   n64Modfil;                  // Last modify of record
    LONGLONG   n64Access;                  // Last Access time
    LONGLONG   n64Allocated;               // Allocated disk space
    LONGLONG   n64RealSize;                // Size of the file
    DWORD      dwFlags;                 // attribute
    DWORD      dwEAsReparsTag;             // Used by EAs and Reparse
    BYTE       chFileNameLength;
    BYTE       chFileNameType;            // 8.3 / Unicode
    WORD       wFilename[512];             // Name (in Unicode ?)

}ATTR_FILENAME;
//////////////////////////////////////////////////////////////////////////////////////
/

#pragma pack(pop, curAlignment)
```

```
class CMFTRecord
{
protected:
    HANDLE m_hDrive;

    BYTE *m_pMFTRecord;
    DWORD m_dwMaxMFTRecSize;
    DWORD m_dwCurPos;
    DWORD m_dwBytesPerCluster;
    LONGLONG m_n64StartPos;

    int ReadRaw(LONGLONG n64LCN, BYTE *chData, DWORD &dwLen);
    int ExtractData(NTFS_ATTRIBUTE ntfsAttr, BYTE *&puchData, DWORD &dwDataLen);

public:
///////// attributes ////////////////////
    ATTR_STANDARD m_attrStandard;
    ATTR_FILENAME m_attrFilename;

    BYTE *m_puchFileData; // collected file data buffer
    DWORD m_dwFileDataSz; // file data size , ie. m_pchFileData buffer length

    bool m_bInUse;

public:
    int SetRecordInfo(LONGLONG n64StartPos, DWORD dwRecSize, DWORD dwBytesPerCluster);
    void SetDriveHandle(HANDLE hDrive);

    int ExtractFile(BYTE *puchMFTBuffer, DWORD dwLen, bool bExcludeData=false);

    CMFTRecord();
    virtual ~CMFTRecord();

};

#endif // !defined(AFX_MFTRECORD_H__04A1B8DF_0EB0_4B72_8587_2703342C5675__INCLUDED_)
```

11.2.8 MFTRecord.cpp

```
// MFTRecord.cpp: implementation of the CMFTRecord class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "MFTRecord.h"


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CMFTRecord::CMFTRecord()
{
    m_hDrive = 0;

    m_dwMaxMFTRecSize = 1023; // usual size
    m_pMFTRecord = 0;
    m_dwCurPos = 0;

    m_puchFileData = 0; // collected file data buffer
    m_dwFileDataSz = 0; // file data size , ie. m_pchFileData buffer length

    memset(&m_attrStandard,0,sizeof(ATTR_STANDARD));
    memset(&m_attrFilename,0,sizeof(ATTR_FILENAME));

    m_bInUse = false;;
}

CMFTRecord::~CMFTRecord()
{
    if(m_puchFileData)
        delete m_puchFileData;
    m_puchFileData = 0;
    m_dwFileDataSz = 0;
}
```

```
// set the drive handle
void CMFTRecord::SetDriveHandle(HANDLE hDrive)
{
    m_hDrive = hDrive;
}

// set the detail
//  n64StartPos is the byte from the starting of the physical disk
//  dwRecSize is the record size in the MFT table
//  dwBytesPerCluster is the bytes per cluster
int CMFTRecord::SetRecordInfo(LONGLONG  n64StartPos, DWORD dwRecSize, DWORD
dwBytesPerCluster)
{
    if(!dwRecSize)
        return ERROR_INVALID_PARAMETER;

    if(dwRecSize%2)
        return ERROR_INVALID_PARAMETER;

    if(!dwBytesPerCluster)
        return ERROR_INVALID_PARAMETER;

    if(dwBytesPerCluster%2)
        return ERROR_INVALID_PARAMETER;

    m_dwMaxMFTRecSize = dwRecSize;
    m_dwBytesPerCluster = dwBytesPerCluster;
    m_n64StartPos =  n64StartPos;
    return ERROR_SUCCESS;
}


/// puchMFTBuffer is the MFT record buffer itself (normally 1024 bytes)
//  dwLen is the MFT record buffer length
//  bExcludeData = if true the file data will not be extracted
//                 This is useful for only file browsing
int CMFTRecord::ExtractFile(BYTE *puchMFTBuffer, DWORD dwLen, bool bExcludeData)
{
    if(m_dwMaxMFTRecSize > dwLen)
```

```
    return ERROR_INVALID_PARAMETER;
if(!puchMFTBuffer)
    return ERROR_INVALID_PARAMETER;

NTFS_MFT_FILE ntfsMFT;
NTFS_ATTRIBUTE   ntfsAttr;

BYTE *puchTmp = 0;
BYTE *uchTmpData =0;
DWORD dwTmpDataLen;
int nRet;

m_pMFTRecord = puchMFTBuffer;
m_dwCurPos = 0;

if(m_puchFileData)
    delete m_puchFileData;
m_puchFileData = 0;
m_dwFileDataSz = 0;

// read the record header in MFT table
memcpy(&ntfsMFT,&m_pMFTRecord[m_dwCurPos],sizeof(NTFS_MFT_FILE));

if(memcmp(ntfsMFT.szSignature,"FILE",4))
    return ERROR_INVALID_PARAMETER; // not the right signature

m_bInUse = (ntfsMFT.wFlags&0x01); //0x01     Record is in use
                              //0x02   Record is a directory

//m_dwCurPos = (ntfsMFT.wFixupOffset + ntfsMFT.wFixupSize*2);
m_dwCurPos = ntfsMFT.wAttribOffset;

do
{   // extract the attribute header

    if(::IsBadReadPtr(&m_pMFTRecord[m_dwCurPos], sizeof(NTFS_ATTRIBUTE)))
    {
        break; // In the debugger this address displays as so don't try to memcpy
    }
```

```
    memcpy(&ntfsAttr,&m_pMFTRecord[m_dwCurPos],sizeof(NTFS_ATTRIBUTE));


switch(ntfsAttr.dwType) // extract the attribute data
{
    // here I haven' implemented the processing of all the attributes.
    //  I have implemented attributes necessary for file & file data extraction
case 0://UNUSED
    break;

case 0x10: //STANDARD_INFORMATION
    uchTmpData = 0;
    nRet = ExtractData(ntfsAttr,uchTmpData,dwTmpDataLen);
    if(nRet)
        return nRet;
    memcpy(&m_attrStandard,uchTmpData,sizeof(ATTR_STANDARD));

    delete uchTmpData;
    uchTmpData = 0;
    dwTmpDataLen = 0;
    break;

case 0x30: //FILE_NAME
    nRet = ExtractData(ntfsAttr,uchTmpData,dwTmpDataLen);
    if(nRet)
        return nRet;
    memcpy(&m_attrFilename,uchTmpData,dwTmpDataLen);

    delete uchTmpData;
    uchTmpData = 0;
    dwTmpDataLen = 0;

    break;

case 0x40: //OBJECT_ID
    break;
case 0x50: //SECURITY_DESCRIPTOR
    break;
```

```
case 0x60: //VOLUME_NAME
    break;
case 0x70: //VOLUME_INFORMATION
    break;
case 0x80: //DATA
    if(!bExcludeData)
    {
        uchTmpData = 0;
        nRet = ExtractData(ntfsAttr,uchTmpData,dwTmpDataLen);
        if(nRet)
            return nRet;

        if(!m_puchFileData)
        {
            try{
                m_dwFileDataSz = dwTmpDataLen;
                m_puchFileData = new BYTE[dwTmpDataLen];
            }
            catch (System::Runtime::InteropServices::SEHException^){
                //unknown error. Thought to be due to mixing of managed and
                unmanaged code
                //return ERROR_INVALID_PARAMETER;
                break;
            }


            if(::IsBadReadPtr(uchTmpData,dwTmpDataLen))
            {
                break; // In the debugger this address displays as so don't try to
                memcpy
            }
            memcpy(m_puchFileData,uchTmpData,dwTmpDataLen);
        }
        else
        {
            puchTmp = new BYTE[m_dwFileDataSz+dwTmpDataLen];
            memcpy(puchTmp,m_puchFileData,m_dwFileDataSz);
            memcpy(puchTmp+m_dwFileDataSz,uchTmpData,dwTmpDataLen);
```

```
            m_dwFileDataSz += dwTmpDataLen;
            delete m_puchFileData;
            m_puchFileData = puchTmp;
        }

        delete uchTmpData;
        uchTmpData = 0;
        dwTmpDataLen = 0;
    }
    break;

case 0x90: //INDEX_ROOT
case 0xa0: //INDEX_ALLOCATION
    // todo: not implemented to read the index mapped records
    return ERROR_SUCCESS;
    continue;
    break;
case 0xb0: //BITMAP
    break;
case 0xc0: //REPARSE_POINT
    break;
case 0xd0: //EA_INFORMATION
    break;
case 0xe0: //EA
    break;
case 0xf0: //PROPERTY_SET
    break;
case 0x100: //LOGGED_UTILITY_STREAM
    break;
case 0x1000: //FIRST_USER_DEFINED_ATTRIBUTE
    break;

case 0xFFFFFFFF: // END
    if(uchTmpData)
        delete uchTmpData;
    uchTmpData = 0;
    dwTmpDataLen = 0;
    return ERROR_SUCCESS;
```

```
        default:
            break;
        };

        m_dwCurPos += ntfsAttr.dwFullLength; // go to the next location of attribute
    }
    while(ntfsAttr.dwFullLength);

    if(uchTmpData)
        delete uchTmpData;
    uchTmpData = 0;
    dwTmpDataLen = 0;
    return ERROR_SUCCESS;
}

// extract the attribute data from the MFT table
//   Data can be Resident & non-resident
int CMFTRecord::ExtractData(NTFS_ATTRIBUTE ntfsAttr, BYTE *&puchData, DWORD &dwDataLen)
{
    DWORD dwCurPos = m_dwCurPos;

    if(!ntfsAttr.uchNonResFlag)
    {// residence attribute, this always resides in the MFT table itself

        puchData = new BYTE[ntfsAttr.Attr.Resident.dwLength];
        dwDataLen = ntfsAttr.Attr.Resident.dwLength;


    memcpy(puchData,&m_pMFTRecord[dwCurPos+ntfsAttr.Attr.Resident.wAttrOffset],dwDataLen);
    }
    else
    {// non-residence attribute, this resides in the other part of the physical drive

        if(!ntfsAttr.Attr.NonResident.n64AllocSize) // i don't know Y, but fails when its
zero
            ntfsAttr.Attr.NonResident.n64AllocSize = (ntfsAttr.Attr.NonResident.n64EndVCN -
ntfsAttr.Attr.NonResident.n64StartVCN) + 1;

        // ATTR_STANDARD size may not be correct
```

```
dwDataLen = ntfsAttr.Attr.NonResident.n64RealSize;

// allocate for reading data
puchData = new BYTE[ntfsAttr.Attr.NonResident.n64AllocSize];

BYTE chLenOffSz; // length & offset sizes
BYTE chLenSz; // length size
BYTE chOffsetSz; // offset size
LONGLONG n64Len, n64Offset; // the actual lenght & offset
LONGLONG n64LCN =0; // the pointer pointing the actual data on a physical disk
BYTE *pTmpBuff = puchData;
int nRet;

dwCurPos += ntfsAttr.Attr.NonResident.wDatarunOffset;;

for(;;)
{
    ///// read the length of LCN/VCN and length //////////////////////
    chLenOffSz = 0;

    memcpy(&chLenOffSz,&m_pMFTRecord[dwCurPos],sizeof(BYTE));

    dwCurPos += sizeof(BYTE);

    if(!chLenOffSz)
        break;

    chLenSz       = chLenOffSz & 0x0F;
    chOffsetSz = (chLenOffSz & 0xF0) >> 4;

    ///// read the data length ////////////////////////////////////////

    n64Len = 0;

    memcpy(&n64Len,&m_pMFTRecord[dwCurPos],chLenSz);

    dwCurPos += chLenSz;

    ///// read the LCN/VCN offset ////////////////////////////////////
```

```
            n64Offset = 0;

            memcpy(&n64Offset,&m_pMFTRecord[dwCurPos],chOffsetSz);

            dwCurPos += chOffsetSz;

            ////// if the last bit of n64Offset is 1 then its -ve so u got to make it -ve
/////
            if((((char*)&n64Offset)[chOffsetSz-1])&0x80)
                for(int i=sizeof(LONGLONG)-1;i>(chOffsetSz-1);i--)
                    ((char*)&n64Offset)[i] = 0xff;

            n64LCN += n64Offset;

            n64Len *= m_dwBytesPerCluster;
            ///// read the actual data ////////////////////////////////////////
            /// since the data is available out side the MFT table, physical drive should
            be accessed
            nRet = ReadRaw(n64LCN,pTmpBuff,(DWORD&)n64Len);
            if(nRet)
                return nRet;

            pTmpBuff += n64Len;
        }
    }
    return ERROR_SUCCESS;
}

// read the data from the physical drive
int CMFTRecord::ReadRaw(LONGLONG n64LCN, BYTE *chData, DWORD &dwLen)
{
    int nRet;

    LARGE_INTEGER n64Pos;

    n64Pos.QuadPart = (n64LCN)*m_dwBytesPerCluster;
    n64Pos.QuadPart += m_n64StartPos;
```

```
    //   data is available in the relative sector from the begining od the drive
    //    so point that data
    nRet = SetFilePointer(m_hDrive,n64Pos.LowPart,&n64Pos.HighPart,FILE_BEGIN);
    if(nRet == 0xFFFFFFFF)
        return GetLastError();

    BYTE *pTmp       = chData;
    DWORD dwBytesRead=0;
    DWORD dwBytes    =0;
    DWORD dwTotRead      =0;

    while(dwTotRead <dwLen)
    {
        // v r reading a cluster at a time
        dwBytesRead = m_dwBytesPerCluster;

        // this can not read partial sectors
        nRet = ReadFile(m_hDrive,pTmp,dwBytesRead,&dwBytes,NULL);
        if(!nRet)
            return GetLastError();

        dwTotRead += dwBytes;
        pTmp += dwBytes;
    }

    dwLen = dwTotRead;

    return ERROR_SUCCESS;
}
```

12. Bibliography

[1] AccessData. *Forensic Toolkit User Guide*. 1.80. http://www.accessdata.com/downloads/media/FTK_1.80_Manual.pdf (accessed April 10, 2010).

[2] Casey, Eoghan. *Handbook of Computer Crime Investigation: Forensic Tools and Technology*. London: Academic Press, 2002.

[3] Davis, Pete. *ThumbDBLib: A C# library for reading thumbs.db files*. http://www.petedavis.net/drupal//index.php?q=node/2 (accessed July 21, 2010).

[4] Goode, Adam. Post to Stack Overflow Web forum, October 21, 2009. http://stackoverflow.com/questions/1557071/the-size-of-a-jpegjfif-image (accessed March 2, 2010).

[5] GreenSpot Technologies Ltd. DM Thumbs. http://www.dmthumbs.com/ (accessed April 10, 2010).

[6] ———. "Photo Recovery and Forensic Software." Detection Master. http://www.detectionmaster.com/ (accessed April 10, 2010).

[7] Guidance Software Inc. "Encase Forensic." Guidance Software. http://www.guidancesoftware.com/computer-forensics-ediscovery-software-digital-evidence.htm (accessed April 10, 2010).

[8] ———. *EnCase Forensic Version 6.11 User's Guide*. 6.11. N.p.: n.p., 2008.

[9] Hamilton, Eric. *JPEG File Interchange Format*. 1.02. http://www.jpeg.org/public/jfif.pdf (accessed March 14, 2010).

[10] Hurlbut, Dustin. *Thumbs DB Files Forensic Issues*. http://www.accessdata.com/media/en_US/print/papers/wp.Thumbs_DB_Files.en_us.pdf (accessed March 14, 2010).

[11] Installsetupconfig. *Master File Table Program Example 1*. http://www.installsetupconfig.com/win32programming/windowsvolumeapis1_19.html (accessed August 14, 2010).

[12] International Telecommunication Union. *Digital Compression and Coding of Continuous-tone Still images*. http://www.w3.org/Graphics/JPEG/itu-t81.pdf (accessed March 14, 2010).

[13] Microsoft. *Windows Compound Binary File Format Specification*. http://msdn.microsoft.com/en-us/library/dd942138.aspx (accessed August 14, 2010).

[14] Rentz, Daniel. *Microsoft Compound Document File Format*. 1.5. http://sc.openoffice.org/compdocfileformat.pdf (accessed July 14, 2010).

[15] Roukine, Michel. Vinetto. http://vinetto.sourceforge.net/ (accessed April 10, 2010).

[16] Schwartz, Martin. *LAOLA file system*. http://user.cs.tu-berlin.de/~schwartz/pmh/guide.html#n1 (accessed June 6, 2010).

[17] Steel, Chad. *Windows Forensics: The Field Guide for Corporate Computer Investigations*. Edited by Carol Long. Indianapolis: Wiley Publishing, Inc., 2006.

[18] Tolmachev, Igor. "Thumbnail Database Viewer." IT Samples. http://www.itsamples.com/thumbnail-database-viewer.html (accessed April 10, 2010).

[19] Vogt, Harald. ThumbsUp. http://people.inf.ethz.ch/hvogt/proj/thumbsup/ (accessed April 10, 2010).

[20] Wiles, Jack, and Anthony Reyes. *The Best Damn Cybercrime and Digital Forensics Book Period*. Edited by Andrew Williams. Burlington: Syngress Publishing, Inc. , 2007.

[21] YogaRamanan, T. Post to www.codeproject.com Web forum, July 9, 2005. http://www.codeproject.com/KB/IP/SNMP_Agent_DLL__Part1_.aspx?msg=1131030#xx1131030xx (accessed August 14, 2010).

[22] ———. *Undelete a file in NTFS*. http://www.codeproject.com/KB/files/NTFSUndelete.aspx (accessed August 14, 2010).