Automation and Parallelization of Forensic Technology for eDiscovery

By

Joel Bowers

A Thesis Presented in Partial Fulfillment of the Requirements for the Degree Master of Information Assurance

CAPS795

Presented to

Dr. Lonnie Decker

2012-07-26
i. Abstract

This paper reports on research done into the requirement, feasibility, development, and testing of prototype tools designed to introduce further automation and parallelization of the forensic technology process as it relates to electronic discovery. Improved use of automation has recently been introduced into the end phases of the Electronic Discovery Reference Model (EDRM). Introducing automation and parallelization at the beginning phases of the EDRM as well should cut down on time and costs associated with the collection and culling of data that is eventually destined for an electronic discovery review platform. All research within is performed with the mindset of preserving the confidentiality, integrity, and accessibility of the information being processed, thus assuring it is still admissible in court as evidence.
**Table of Contents**

i. Abstract .......................................................................................................................... 2  
Table of Contents .................................................................................................................. 3  
Table of Figures .................................................................................................................... 4  
1. Introduction ...................................................................................................................... 5  
2. Literature Review ............................................................................................................. 7  
  Forensic Technology in Recent Years ............................................................................... 7  
  Current Forensic Technology Challenges ........................................................................... 9  
  Digital Forensics as it Relates to Electronic Discovery .................................................... 12  
  Feasibility of Automating the Digital Forensic Electronic Discovery Process .................. 15  
3. Methods Used to Gather Information ............................................................................. 19  
  Development of a Parallel Imaging Proof of Concept ..................................................... 19  
  Development of an Automated Extraction and Culling Proof of Concept ...................... 22  
  Development of an Automated Production Proof of Concept ........................................ 23  
4. Results ................................................................................................................................ 25  
  Parallelized Collection Proof of Concept Results ............................................................ 25  
  Automated Extraction and Culling Proof of Concept Results ......................................... 28  
  Automated Production Proof of Concept Results ............................................................. 29  
5. Conclusions ...................................................................................................................... 34  
Appendices ........................................................................................................................... 37  
  A – doforensics.sh .............................................................................................................. 37  
  B.1 – DiscoveryAutomator_Lib.EnScript .......................................................................... 39  
  B.2 – DiscoveryAutomator.EnScript .................................................................................. 43  
  C – dtSearchAutomator.java ............................................................................................. 47  
References ............................................................................................................................ 49
## Table of Figures

Figure 1 - W3Counter Global Web Stats (W3Counter, 2012) ........................................... 8  
Figure 2 – Results of Computer Forensics Needs Survey (Rogers, Siegfried, 2004) ..11  
Figure 3 – Electronic Discovery Reference Model (EDRM, 2009) ............................ 13  
Figure 4 - Tableau T35i Forensic Bridge (Digital Intelligence, 2012) ........................... 20  
Figure 5 – Parallelized Forensic Imaging (Davies, 2012) ............................................. 27  
Figure 6 - BeyondCompare Output (Bowers, 2012) ..................................................... 28  
Figure 7 - dtSearchAutomator Test Document Set (Bowers, 2012) ........................... 30
1. Introduction

Computer technology is steadily becoming a larger part of people’s lives every day, and is used in some way for almost everything anyone does in a business environment. Because of the increasing presence of these machines, and the corresponding increasing amount of fraudulent behavior involving computers, the use of forensic technology and electronic discovery to assist with investigations and litigation is also increasing steadily. A report by Gibson Dunn in July 2009 concurs.

"From the upswing in sanctions to new constitutional questions, e-discovery continues to be an important and rapidly developing area of law. While several major themes characterize this year’s opinions thus far, their enduring lesson is that litigants and their counsel need to adapt in a period of rapid technological and legal change." (Dunn, 2009)

Ironically, the rapid technological and legal change that Dunn wrote about is the reason that additional technological and legal change is required; in this case to satisfy the growing demand for forensic technology and electronic discovery services. Historically, the best solution to satisfy a higher demand for a product has been to mass-produce the product in an assembly line or industrialized fashion. In more recent years, innovations in standardization and automation have increased the production capacity, as well as the "build quality" of the products that they are applied to. After all, one of the best ways to avoid human error is to not get a human involved in a process except at points where critical thinking is required.

This paper reports on research conducted into the feasibility of the automation and parallelization of Forensic Technology techniques as they relate to
the electronic discovery process. Since many of the same tasks are completed during the forensic technology portion of every electronic discovery job, it is expected that much of the work that goes into the forensic technology portion of an electronic discovery production can be automated. Further, it is expected that this automation will reduce the overall man-hours spent while increasing the quality of the final productions seen in court. Above all, the research conducted focuses on assuring that the electronically sourced information (ESI) being processed is kept confidential, while ensuring that the integrity of any ESI being processed is maintained, and that accessibility of the ESI by the required parties is preserved. As a side effect of this research, it is expected that various important statistical "milestones" that are frequently discussed during forensic technology and electronic discovery processing will be generated and presented to the appropriate parties. The generation of statistical milestone reports means that the benefits of any required critical thinking can still be applied to the process at any time.
2. Literature Review

Forensic Technology in Recent Years

Simson Garfinkel provides a great overview of forensic technology over the past decade in his article "Digital forensics research: The next 10 years". The following excerpt summarizes this overview.

"The years from 1999-2007 were a kind of “Golden Age” for digital forensics. During this time digital forensics became a kind of magic window that could see into the past (through the recovery of residual data that was thought to have been deleted) and into the criminal mind (through the recovery of email and instant messages). Network and memory forensics made it possible to freeze time and observe crimes as they were being committed even many months after the fact. Forensics became so widespread and reliable that it escaped from the lab and onto the TV screen, creating the so-called “CSI Effect.” (Shelton, 2008)

This Golden Age was characterized by:
- The widespread use of Microsoft Windows, and specifically Windows XP.
- Relatively few file formats of forensic interest, mostly Microsoft Office for documents, JPEG for digital photographs; and AVI and WMV for video.
- Examinations largely confined to a single computer system belonging to the subject of the investigation.
- Storage devices equipped with standard interfaces (IDE/ATA), attached using removable cables and connectors, and secured with removable screws.
- Multiple vendors selling tools that were reasonably good at recovering allocated and deleted files." (Garfinkel, 2010)

Everything listed in Garfinkel's above description of the "Golden Age" of digital forensics is changing, or is about to change. Microsoft Windows is seeing a decline in use, and Windows XP is finally being retired by many users and corporate Information Technology departments. Windows’ decline is augmented by the rise of smartphones and tablets as replacements for typical PCs. There are more file types in use now than ever, and it is difficult to nail down a specific list of extensions that
fits every investigation. The below chart from W3counter shows that Windows 7 is now more popular than Windows XP, and that alternative operating systems such as Mac OS X and tablet operating systems such as iOS and Android are gaining market share.

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windows 7</td>
</tr>
<tr>
<td>2</td>
<td>Windows XP</td>
</tr>
<tr>
<td>3</td>
<td>Apple OS X</td>
</tr>
<tr>
<td>4</td>
<td>Windows Vista</td>
</tr>
<tr>
<td>5</td>
<td>Apple iOS</td>
</tr>
<tr>
<td>6</td>
<td>Android</td>
</tr>
<tr>
<td>7</td>
<td>Linux</td>
</tr>
<tr>
<td>8</td>
<td>BlackBerry</td>
</tr>
<tr>
<td>9</td>
<td>SymbianOS</td>
</tr>
<tr>
<td>10</td>
<td>Windows 8</td>
</tr>
</tbody>
</table>

Figure 1 - W3Counter Global Web Stats (W3Counter, 2012)

Modern investigations and litigations are rarely focused on only one piece of ESI. Even if there is only one technology user involved in a case, that user will still likely have a desktop PC and/or a laptop, as well as a smartphone and a tablet at the bare minimum. Further, there are usually shared resources involved such as remote servers and external media that the user being investigated may use on a daily basis. Storage devices are often times not equipped with standard interfaces. One example
of a popular device that does not have a traditional hard drive or solid-state drive with a standard interface is Apple's MacBook Air. Products like the MacBook Air and the iPhone are not designed to be user serviceable, or even opened up by the end user, as they have very few replaceable parts. This poses a great challenge for forensically imaging (making an exact bit for bit copy that is defensible in court) of these devices. Garfinkel’s final point about the "Golden Age" references the ability of industry standard software for recovering allocated and deleted files. This is also something that is more difficult today because of the widespread use of flash memory, as well as alternative file systems. NTFS and FAT are no longer the only file systems that have to be dealt with by forensic technology teams anymore.

**Current Forensic Technology Challenges**

The largest upcoming challenge in forensic technology as it relates to electronic discovery is the sheer volume of data that currently resides on information systems. The paper "Digital Forensics: Highlighting A Research Agenda" outlines this in more detail, and proposes parallelization as a potential solution to be researched to help solve this problem.

"It is common for digital forensic investigations to be overwhelmed with massive volumes of data. Increasing numbers of devices hold potentially relevant information, and the data storage capacity on such devices is expanding rapidly. It is easy to find examples of digital media players with 160GB hard drives, inexpensive digital cameras that can store 8GB or more, cell phones that have 16GB of flash storage, inexpensive 8GB USB memory sticks, and consumer-grade terabyte hard disks costing no more than a few hundred dollars. In addition, dedicated storage devices offer almost limitless storage volumes, and while Storage Area Network (SAN) devices still tend to be limited to larger corporate environments, consumer-level Network Attached Storage (NAS) devices are available at
prices that make them practical for home and small office environments. All of this means that a typical investigation can involve massive volumes of data. While some effort has been made towards parallelization of data processing (such as Access Data’s Distributed Network Attack product that parallelizes password recovery across multiple workstations), much more remains to be done if useful information is to be retrieved from these increasingly common large data collections. Areas identified by the group in which parallelization research could provide benefits included traffic generation, the imaging and carving processes, and the development of user history timelines, including those based on multiple data sources. In addition, approaches that combine data imaging and evidence identification in parallel could also be beneficial, allowing an investigator to potentially direct the data acquisition process based on real-time results to acquire the most promising data sources during the initial phase of analysis.” (Nance, Hay, Bishop)

In addition to the suggestions about parallelizing the imaging and carving processes (extracting files from a filesystem) on multiple sources by Nance, Hay and Bishop above, a study analyzed by Rogers and Siegfried in 2004 surveyed a large sample set of computer forensic investigators and asked them to rank the emerging problems and challenges in the field as experienced by those investigators. The study was described as follows.

"A single question, free form answer, survey was posted on the forensics research webpage belonging to the Center for Education and Research in Information Assurance and Security (CERIAS), Purdue University. The survey was identified as being informal and simply an attempt to begin collecting meaningful data for the computer forensics community. The survey consisted of a single question that asked the respondents to list what they considered to be the top five issues related to computer forensics. Once the respondents filled out the drop down box, the response was anonymously e-mailed to a generic e-mail account belonging to the principal researcher. The duration of the study was approximately one month. The existence of the web survey was made known via postings to the various news list-servers dedicated to computer forensics (e.g., computer forensic tool testing, computer forensic investigator, forensics, Internet crime, and Linux forensics).” (Rogers, Siegfried, 2004)

Below is the graph showing the results of the survey.
Tools, Data Acquisition, and Technologies were all in the top 5 issues identified by the survey, with more than 15% of respondents identifying these areas as areas that needed improvement. The research being conducted in this paper is aimed at addressing those three concerns as they converge. All prototypes discussed in the research portion below, including the suggested approach for data acquisition, will aim to lay the foundation for new tools that are designed with parallelization and efficiency in mind, and are technology agnostic where possible.
Keeping the tools technologically agnostic ensures the tools can be applied to, and are effective at the acquisition and processing of data from as many different pieces of ESI as possible.

**Digital Forensics as it Relates to Electronic Discovery**

Digital forensics as a discipline is a branch of computer science. The end goal of any digital forensic process is examining electronically stored information, and then correlating and interpreting that information so that facts relating to a particular matter can be admitted into court as evidence. For these facts to be admitted as evidence, there must be a scientifically proven method for preserving and analyzing the information in question so that the integrity of those facts cannot be disputed by opposing counsel. The main computer science method that forensic technology relies on is the “hashing” of files. In computer science, hashing is the act of running a predetermined mathematical algorithm against the entire contents of a particular file in order to produce a string of characters that represents that file and only that file, similar to a digital fingerprint.

Electronic discovery relies on the same principles as digital forensics in that it uses the same computer science methodologies to prove that all the responsive documents’ integrity has been properly maintained, but usually applying to a much larger amount of information. When collecting and processing documents, a professional must be very meticulous in documenting the collection and ensuring that all the steps are followed to maintain the integrity of those documents. Since so
much attention to detail is required, and the volume of information is so large, it often takes digital forensic professionals a large amount of time to prepare the documents for use in court. These procedures can end up being quite expensive, as confirmed by Foss below.

"Despite the efforts in recent years to reduce the cost of discovery, eighty-five percent of attorneys believe that discovery is still too expensive, according to a recent study. This Note discusses two main reasons why e-discovery is expensive—document review and collection—and urges courts to implement certain discovery rules in a way that will incentivize the utilization of technological advancements that could cut down on costs. In considering (1) whether document reviews for relevancy and privilege are adequate, or (2) whether to compel production of ESI that is extremely expensive and burdensome to collect, courts should consider the technological capabilities of the parties and hold them to standards of reasonableness. Applying standards of reasonableness to both document review and pre-litigation management of ESI ultimately will help to reduce e-discovery costs by stimulating innovation in the e-discovery and information-system-management software industries." (Foss, 2012)

To illustrate the overlap of the forensic technology and electronic discovery processes, the figure below depicts the Electronic Discovery Reference Model.

![Electronic Discovery Reference Model](image-url)
Forensic technology as it is discussed in this paper refers to the **Preservation, Collection and Processing** portions of the Electronic Discovery Reference Model shown above.

In the field of digital forensics and electronic discovery, preservation is simply the process of ensuring that all the potentially relevant data is not destroyed. This could mean removing hard drives from computers and replacing them with new ones, pulling backup tapes out of rotation, or performing special backups of certain media. Any steps that could be taken to ensure that potentially relevant data is not being overwritten are counted as part of the preservation phase. Media that is preserved in this way is considered original evidence, and is never worked on directly – forensically sound copies of the data, called forensic images, are created first. The forensic images act as working copies for investigators, so that the original evidence may be preserved and remain untouched.

Collection is the process of actually imaging or restoring data from the media that has been preserved in the previous preservation phase. During the collection phase, a second copy of all the preserved data is collected onto a repository for working data. The working data repository is then used as input for the next phase, processing.

In electronic discovery terms, processing is the label used for culling down the initially collected data set of files that are likely not responsive, and then
Feasibility of Automating the Digital Forensic Electronic Discovery Process

Daniel Ayers wrote the following in his 2009 paper “A second generation computer forensic analysis system”.

"Repeatability and robustness is improved if the detailed sequencing of analysis tasks is automated, following an investigation plan set by the analyst or as a matter of policy in the computer forensic laboratory. EnCase has limited support for automated execution via the EnScript language and scripts, such as the Case Processor, supplied with the software. Analysts are free to develop and distribute their own scripts. But scripting is a partial solution at best as there are many EnCase functions that cannot be automated in scripts. Furthermore, the EnScript documentation is sparse and does not provide sufficient detail for an analyst to develop scripts and be confident that the results generated will be as expected." (Ayers, 2009)

Ayers recognizes the value of automation, but is not being optimistic enough in his assessment concerning the feasibility of automation in the forensic analysis process. The scripting language that Ayers mentions, EnScript, is the scripting language built into Guidance Software’s EnCase, an industry standard piece of forensic analysis software. EnScript has a feature that will allow the analyst to execute external binaries, and will accept output from those binaries after they have completed running as a return statement. Anything that cannot be handled by EnScript itself could easily be implemented in another programming language and then installed into the EnScript directory with the script. This procedure would
remove any limitations that may have been present on what can be accomplished or what can be automated inside EnCase, enabling almost anything to be automated.

While automation is technically possible, there is still the matter of getting a judge to agree that it is a good idea.

Fortunately, automation of the Review phase of the EDRM has been recently gaining support with a technique known as predictive coding. Predictive coding is a new technology that is maturing very rapidly. To briefly describe the process, a team of lawyers will review and mark around 10% of a data set with any issue tags relevant to the case. The 10% is provided to the lawyers by random sampling from all documents and document types. When the review of this initial 10% of the data set is complete, the predictive coding software then uses a technique called latent semantic indexing to create an advanced mathematical model of the document set that sorts the documents thousands of different ways. An excerpt from the abstract of the paper “Latent Semantic Indexing: A Probabilistic Analysis” explains the concept briefly:

“Latent semantic indexing (LSI) is an information retrieval technique based on the spectral analysis of the term-document matrix, whose empirical success had heretofore been without rigorous prediction and explanation. We prove that, under certain conditions, LSI does succeed in capturing the underlying semantics of the corpus and achieves improved retrieval performance. We also propose the technique of random projection as a way of speeding up LSI. We complement our theorems with encouraging experimental results.” (Papadimitriou, Raghavan, Tamaki, Vempala, 1998)

After the initial set is complete and the latent semantic index is generated, the computer then uses the scores of how well the documents fit into each of the
various issue codes and then predicts which of the remaining documents fall under which code, eliminating the need for humans to complete the review themselves.

On April 23, 2012, James H. Chamblin, a Circuit Judge in Loudoun County Virginia executed the following order approving the use of predictive coding in the case of Global Aerospace Inc. vs. Landow Aviation.

"Having heard argument with regard to the Motion of Landow Aviation Limited Partnership, Landow Aviation 1, Inc., and Landow & Company Builders, Inc., pursuant to Virginia Rules of Supreme Court 4:1(b) and (c) and 4:15, it is hereby ordered Defendants shall be allowed to proceed with the use of predictive coding for purposes of the processing and production of electronically stored information, with processing to be completed within 60 days and production to follow as soon as practicable and in no more than 60 days. This is without prejudice to a receiving party raising with the court an issue as to completeness or the contents of the production or the ongoing use of predictive coding." (Chamblin, 2012)

An additional example of a judge ordering the use of predictive coding, is the Honorable Andrew Jay Peck, a United States Magistrate Judge from the state of New York. Judge Peck ordered the following use of predictive coding on February 24, 2012 in the high profile Da Silva Moore vs. Publicis Group case.

"The parties have discussed the methodologies or protocols for the search and review of ESI collected from the EMC SourceOne archive and the following is a summary of the Parties' agreement on the use of Predictive Coding. This section relates solely to the EMC SourceOne data source (hereinafter referred to as the "e-mail collection"). MSL will utilize the Axcelerate software by Recommind to search and review the e-mail collection for production in this case. The process begins with Jackson Lewis attorneys developing an understanding of the entire e-mail collection while identifying a small number of documents, the initial seed set, that is representative of the categories to be reviewed and coded (relevance, privilege, issue-relation). It is the step when the first seed sets are generated which is done by use of search and analytical tools, including keyword, Boolean, and concept search, concept grouping, and as needed, up to 40 other automatically populated filters available within
the Axcelerate system. This assists in the attorney's identification of probative documents for each category to be reviewed and coded..." (Peck, 2012)

In light of these recent developments in the review phase combined with the potential benefits that could be gained by improving efficiency of the digital forensics side of the electronic discovery process, it makes sense to conduct research into what could be done to increase the use of automation and parallelization of forensic technology as it applies to electronic discovery.
3. Methods Used to Gather Information

“The principle of science, the definition, almost, is the following: the test of all knowledge is experiment. Experiment is the sole judge of scientific "truth". But what is the source of knowledge? Where do the laws that are to be tested come from? Experiment, itself, helps to produce these laws, in the sense that it gives us hints. But also needed is imagination to create from these hints the great generalizations—to guess at the wonderful, simple, but very strange patterns beneath them all, and then to experiment to check again whether we have made the right guess.” (Feynman, 1961)

The literary review section of this paper showed that there is a growing industry need for more efficient, modernized forensic technology tools. The research section of this paper will propose some potential solutions and then validate or invalidate those solutions through scientific experimentation. For each of the proposed areas of improvement, a proof of concept will be developed. Then, those proofs of concept will be compared against the results derived from traditional manual methods. The goal is to show that implementing parallelization and automation solutions into the forensic technology portion of the electronic discovery process will improve accuracy while reducing the overall time investment required.

Development of a Parallel Imaging Proof of Concept

Traditionally, a physical hardware write blocker like the one pictured below has always been required to create a forensically sound image of a computer’s hard drive.
Hardware write blocking devices (devices that physically prevent data being written to a hard drive) work almost flawlessly and are extremely reliable for small investigations. The problem with hardware write blocking devices is, if a discovery is taking place at a large company with 300 or more images to be acquired, it could take weeks just to complete the imaging, depending on the number of hardware write blockers available.

Using a Linux-based boot disc to start up the computers in a read only mode, and then using the “dd” utility to create the bit-stream images could enable the imaging process to be parallelized in such a way that there is no limit to the number of computers that could be imaged simultaneously. “dd” is an old Unix utility that can be used to make forensic images of disks. The proof of concept will consist of a Linux based boot disc containing all the software necessary to create a bit-stream image, as well as modifications to prevent the system from automatically writing anything to the discs inside the computer. The process can be validated by comparing the MD5 hash of a bit stream image generated by the traditional
hardware write blocking method, against the hash value generated while acquiring
the image with the boot disc method. If the hashes match, then it proves that the
software write blocking method was able to reproduce the exact same results as the
hardware write blocker.

The proposed proof of concept is based on the portable Slax Linux
distribution. To obtain the bit stream for the image, it will use “dc3dd”; a patched
version of GNU dd with added features designed for use in computer forensics.
“dc3dd” was developed by Jesse Kornblum at the US Department of Defense Cyber
Crime Center. Jesse’s patch added the following features to dd:

- On the fly hashing with multiple algorithms
- Able to write errors directly to a file
- Combined error log
- Pattern wiping
- Verify mode
- Progress reports
- Split output
- “Zeroing out” of bad sectors

To encapsulate the image into the EnCase Expert Witness Format (EWF files, the
proprietary forensic image format used by EnCase), the following proof of concept
will use Joachim Metz’ “ewfacquirestream” software, part of the “libewf” package.
“libewf” is a library that allows a forensic analyst to work with EWF files inside a
Unix environment. “ewfverify”, another piece of software included in the “libewf”
package, will be used to verify that the images were created successfully.
Development of an Automated Extraction and Culling Proof of Concept

After collection, the next phase of an electronic discovery engagement is the processing and analysis of the collected data. On the digital forensics side, this usually entails:

- Verifying the disk images
- Recovering any deleted files
- Completing a file signature analysis
- Filtering documents by extension
- Filtering documents by date range
- Filtering documents by keyword
- Copying the data out while preserving metadata

To begin automating this process, a consolidated GUI window with the following options would be presented to the user upon running the EnScript.

- File Chooser: Path to a hard drive image in Expert Witness format
- Checkbox: Recover Deleted Files?
- Date Range: Date Filter Start Date
- Select Box: Date Filter Start Date Type (Modified, Accessed, Created)
- Date Range: Date Filter End Date
- Select Box: Date Filter End Date Type (Modified, Accessed, Created)
- Textbox: Loose File Extensions to Export
- Textbox: Mail Extensions to Export
- Textbox: Keywords to Search
- File Chooser: Output Location

After the user fills out the form, the script should:

1. Add the evidence file to the case
2. Wait for verification to complete
3. Run "Recover Lost Folders" on all partitions if the checkbox is checked
4. Add the keyword list to the case, and "blue check" those keywords
5. Run a signature analysis on all the files with matching extensions
6. “Blue check” all the files with the following conditions:
   - Signature matches, or mismatches of a type in the extension list
   - Matches a requested extension
   - Matches the date range
7. Perform a "Search" for all blue checked keywords on all "blue check" files
8. “Un-bluechecks” all files
9. “Blue checks” all files with search hits, all mail files with signature matches
10. “Export Folders...” on all selected files to the specified output location
Developing the prototype described above would have taken too long if written autonomously, so assistance was requested from Professor Nick Johnston of the Sheridan Institute of Technology. Professor Johnston graciously accepted, and was able to develop a prototype version of the DiscoveryAutomator EnScript seen in Appendix B of this paper.

For the experiment, the results from a DiscoveryAutomator test case will be compared against the results of performing the process manually using BeyondCompare. If the resulting output sets are the same, then the experiment will be considered successful for the test case. Of course, extensive testing would be required to put the prototype into a production environment.

**Development of an Automated Production Proof of Concept**

After forensic technology processing, the culled data would then be passed off to electronic discovery personnel. First, the forensic technology team will have to determine which files to transfer. This step is often completed with a tool called DtSearch that creates an index of all the words found in a specified set of files, and then allows searches to be run on the index. Finally, the forensic technology team would need to ensure that any metadata is protected while the data is being transferred. A regular “Windows copy” would not do this, so a tool like Robocopy would have to be used. Robocopy is a tool that is built into current versions of Windows that allows a user to copy files from one device to another while preserving the metadata of the files being copied. Robocopy does allow some automation by default, but the level of automation could still be improved when it
comes to copying out files that are identified when running searches, or even generating batch files of things that need to be Robocopied.

The proof of concept for automating dtSearch and Robocopy will involve reading in a keyword list from a text file, automatically generating a report showing the hits and data size for those keywords, and then generating a batch file that can be used to run Robocopy to copy those search results while preserving the metadata. dtSearch has a comprehensive Java API that allows automation of the application. The prototype for this tool is attached to this paper as Appendix C.

The Robocopy switch needed to preserve the metadata of the files is “/COPY:DAT” which copies Data, Attributes and Timestamps.

Similarly to the previous processing experiment, this experiment will be considered a success for the test case if executing the manual process provides the same results as executing the prototype.
4. Results

Parallelized Collection Proof of Concept Results

**Hypothesis:** Imaging with a write blocker will produce a bit-stream image with the same MD5 hash value as imaging the same hard drive with the “doforensics” script as specified in Appendix A.

**Method:** Image the hard drive with the write blocker, and note the MD5 hash. Then image the hard drive with “doforensics”, and note the MD5 hash. Compare the MD5 hashes.

**Observations:**

doforensics / Software Write Blocker Output:

```
ewfacquirestream 20100226 (libewf 20100226, libuna 20091031, libbfio 20091114, zlib 1.2.3, libcrypto 0.9.8, libuuid)

Using the following acquisition parameters:
Image path and filename: /mnt/ImageFiles/Laptop_Kit-HDD5-20120705/Laptop_Kit-HDD5-20120705.E01
Case number: thesis:Laptop_Kit-HDD5-20120705
Description: Laptop_Kit-HDD5-20120705
Evidence number:
Examiner name: Bowers
Notes:
Media type: fixed disk
Is physical: yes
Compression used: none
EWF file format: EnCase 6
Acquire start offset: 0
Amount of bytes to acquire: 0 (until end of input)
Evidence segment file size: 1.8 GiB (2000000000 bytes)
Bytes per sector: 512
Block size: 128 sectors
Error granularity: 64 sectors
Retries on read error: 2
Wipe sectors on read error: yes

MD5 hash stored in file: 114a2097c010baf38d5e72cea2bcffa0
MD5 hash calculated over data: 114a2097c010baf38d5e72cea2bcffa0
```
ewfverify: SUCCESS

/dev/sda:

Model=HITACHI HTS541660J9SA00, FwRev=SBBIC7JP, SerialNo=SB2B41SLGJP0EE
Config={ HardSect NotMFM HdSw>15uSec Fixed DTR>10Mbs }
RawCHS=16383/16/63, TrkSize=0, SectSize=0, ECCbytes=4
BuffType=DualPortCache, BuffSize=7516kB, MaxMultSect=16, MultSect=?16?
CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=117210240
IORDY=on/off, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
PIO modes: pio0 pio1 pio2 pio3 pio4
DMA modes: mdma0 mdma1 mdma2
UDMA modes: udma0 udma1 udma2 udma3 udma4 *udma5
AdvancedPM=yes: mode=0x80 (128) WriteCache=enabled
Drive conforms to: ATA/ATAPI-7 T13 IS32D revision 1: ATA/ATAPI-2,3,4,5,6,7

* signifies the current active mode

FTK Imager / Physical Write Blocker Output

Created By AccessData® FTK® Imager 3.1.0.1514

Case Information:
Acquired using: ADI3.1.0.1514
Case Number: Thesis
Evidence Number: 
Unique Description: 
Examiner: JDB
Notes:

-------------------------------------------------------------------------------------------------------------------------------------

Information for F:\ImageFiles\Kit-Physical:

Physical Evidentiary Item (Source) Information:

[Drive Geometry]
Cylinders: 7,296
Tracks per Cylinder: 255
Sectors per Track: 63
Bytes per Sector: 512
Sector Count: 117,210,240

[Physical Drive Information]
Drive Model: HT554106 009SA00 USB Device
Drive Serial Number: MPBCPAXMGSSLG
Drive Interface Type: USB
Source data size: 57231 MB
Sector count: 117210240

ATTENTION:
The following sector(s) on the source drive could not be read:
65446562

The contents of these sectors were replaced with zeros in the image.
Conclusion: Since the image acquired by traditional means resulted in the same MD5 hash (114a2097c010baf38d5e72cea2bcffa0) as the image acquired by “doforensics”, even on an image of a hard disk containing a bad sector, this shows that “doforensics” is an extremely effective tool that could be used for the parallelization of bit-stream imaging, with zero investment in additional software or hardware.
Automated Extraction and Culling Proof of Concept Results

**Hypothesis:** Running DiscoveryAutomator should produce the same output as performing the process manually, but it should do so with no downtime waiting for input from humans.

**Method:** Perform the process manually, as well as with DiscoveryAutomator, while timing both. Then, run BeyondCompare on the data sets to see if the same results were reached.

**Observations:** Running DiscoveryAutomator produced the exact same output as performing the process manually. The below BeyondCompare screenshot shows the results of the comparison which confirms those results.

![BeyondCompare Output](image)

*Figure 6 - BeyondCompare Output (Bowers, 2012)*
**Conclusions**: The ability to tell the computer up front all the steps that will be necessary for the processing phase of a discovery, and then walking away to work on other things, or simply moving on to the next piece of electronic evidence in the case would be invaluable. If this script was developed further it could rival the more robust versions of EnCase such as EnCase eDiscovery.

**Automated Production Proof of Concept Results**

**Hypothesis**: Running dtSearchAutomator will provide the same results as performing the searches manually, and it will do so in a much shorter timeframe.

**Method**: Perform the search manually as well as with the dtSearchAutomator software. Then, compare the output numbers of the search results against each other.
The data set to be searched is the following set of 10 text documents:

Figure 7 - dtSearchAutomator Test Document Set (Bowers, 2012)

The search terms to be tested are as follows:
1. cow
2. cow OR sheep
3. cow w/1 (duck OR chicken)
4. (cow AND chicken) AND NOT duck
5. NOT chicken
6. \((\text{pig OR chicken}) \text{ AND duck}\)
7. \(\text{sheep AND NOT pig}\)
8. \(\text{pig OR tyrannosaurus}\)
9. \(\text{NOT tyrannosaurus}\)
10. \(\text{pig AND sheep AND cow AND duck}\)

Observations:

The Manual dtSearch Output (dtSearch GUI does not provide file size):

cow: 5
cow OR sheep: 7
cow w/1 (duck OR chicken): 5
(cow AND chicken) AND NOT duck: 1
NOT chicken: 5
(pig OR chicken) AND duck: 5
sheep AND NOT pig: 1
pig OR tyrannosaurus: 6
NOT tyrannosaurus: 9
pig AND sheep AND cow AND duck: 2

The dtSearchAutomator Output:

C:\Users\jbowers\Desktop\Desktop\dtSearchAutomator\dist>java -jar dtSearchAutomator.jar stats testindex keywords.txt
cow: 5, 1621 bytes
cow OR sheep: 7, 2253 bytes
cow w/1 (duck OR chicken): 5, 1621 bytes
(cow AND chicken) AND NOT duck: 1, 315 bytes
NOT chicken: 5, 1582 bytes
(pig OR chicken) AND duck: 5, 1625 bytes
sheep AND NOT pig: 1, 328 bytes
pig OR tyrannosaurus: 6, 1915 bytes
NOT tyrannosaurus: 9, 2869 bytes
pig AND sheep AND cow AND duck: 2, 657 bytes

The dtSearchAutomator “Export Mode” output:

C:\Users\jbowers\Desktop\Desktop\dtSearchAutomator\dist>java -jar dtSearchAutomator.jar export testindex keywords.txt
rocopcopy "testdocs" "exportdocs" 08.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 07.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 06.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 05.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 04.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 03.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 02.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 07.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 06.rtf /COPY:DAT
rocopcopy "testdocs" "exportdocs" 05.rtf /COPY:DAT
Conclusions:

Both searches produced the same output as expected. The Robocopy export batch script to copy the files over to the production location can contain duplicates if there are multiple keywords in the input file at the moment. It still works in practicality since robocopy will not copy a file that is the “same” as one in it’s target directory, it just moves on. The real advantage to this piece of automation is going to be the preliminary results for searches pre-export. In the past, a person would have to sit there and manually type or paste in each query to see the differences in hit counts. Those results were typically used in order to improve the keywords effectiveness, mainly by removing keywords that provided too many false positives. Automating this type of report will allow a user to just type in all possible keywords or
combinations of keywords they would like to see, and then be provided with the output without working around the dtSearch GUI.
5. Conclusions

The literary review at the beginning of this paper showed that due to an increasing amount of ESI, ESI increasing in storage size, and an increasing reliance on ESI as evidence in investigations and litigations, that forensic technology practices and tools need modernization in order to keep up with the growing demand.

The research in this paper focused on developing prototypes of tools to automate and parallelize the forensic technology processes as they relate to electronic discovery services, and then experiments were performed in order to validate that the prototypes performed the tasks just as accurately as the manual traditional methods.

Three prototypes were developed for the paper. The first, doforensics, is a script designed to remove the need for hardware write blocking and to allow a collection team to completely parallelize all required image creation. The second, DiscoveryAutomator, is an EnScript designed to automate all the manual tasks that usually had to be performed in EnCase to prepare data from image files for electronic discovery. The final prototype was a java application that used the dtSearch API to automate the generation of statistics for keyword preparation, as well as to create a Robocopy batch script to copy the required files out of the main dataset to be used in electronic discovery.
The experiments used to validate the results of all three of the proofs of concept applications were successful. In each case, the automated tools were able to exactly reproduce the results of all the manual methodologies, and were able to do so with greatly reduced input from a human, thus reducing the potential for human error. In addition, all three tools were designed to ensure the confidentiality, integrity, and accessibility of all the electronically stored information that was processed.

While the prototypes developed here show that the concepts are viable, there is still a requirement for further research and development before the tools can be put into production.

For doforensics, there are plans to add the following features:
- Custom kernel with support for more chipsets and more modern hardware
- Ability to image multiple hard drives from one machine at the same time
- TrueCrypt functionality, to protect image files and for target drive recognition
- Drive wiping functionality

For DiscoveryAutomator, there are plans to add the following features:
- Deduplication for files and e-mail
- The signature analysis feature does not work yet
- The ability to call dtSearchAutomator from the EnScript

For dtSearchAutomator, there are plans to add the following features:
- More robust reporting
- Automated keyword patternning and statistics (try all combinations of keywords and provide counts)

In addition to the planned features, all tools will also need to be thoroughly tested before they are put into production environments. The appendices below
contain the source code for all the scripts and applications developed for the proofs of concept.
Appendices

A – doforensics.sh

#!/bin/bash
#########################################################
# Program: doforensics
# Purpose: To do forensics
# Author: Joel Bowers (joel@bowe.rs)
# Version: 1.2
#########################################################

echo -n "Welcome to forenslax 1.2, please wait a minute while I show you some disks."

lsbsh -quiet -short -class disk -class volume

echo "The next two questions are the most important questions. If you mix them up, you will OVERWRITE data and DESTROY evidence."

echo -n "From the list above, please enter the path to the DEVICE you would like to image: [/dev/sda] 
read -e SOURCE
if [ -z "$SOURCE" ]; then
  SOURCE="/dev/sda"
fi

echo -n "From the list above, please enter the path to the PARTITION you would like to OUTPUT the image to: [/dev/sdb1] 
read -e TARGET
if [ -z "$TARGET" ]; then
  TARGET="/dev/sdb1"
fi

echo "OK, you better be right, because I'm going to look and see if you told me to overwrite a custodian's hard drive now.");

mount -o rw -t ntfs-3g "$TARGET" /mnt

if [ -d "/mnt/WINDOWS" ]; then
  echo -n "If you can read this you are really bad at forensics."
  exit 1
fi

mount -o rw -t ntfs-3g "$TARGET" /mnt

if [ ! -d /mnt/ImageFiles ]; then
  mkdir /mnt/ImageFiles
fi

mount -o rw -t ntfs-3g "$TARGET" /mnt

if [ ! -d /mnt/ImageFiles ]; then
  mkdir /mnt/ImageFiles
fi

echo " "
echo ""  
echo -n "Please enter the required parameters."

echo ""  
read -e CASENUMBER  

echo -n "Case number: "
read -e CASENUMBER  

echo -n "Investigator's First Name: "
read -e IFIRSTNAME  

echo -n "Investigator's Last Name: "
read -e ILASTNAME  

echo -n "Custodian's First Name: "
read -e CFIRSTNAME  

echo -n "Custodian's Last Name: "
read -e CLASTNAME  

echo -n "How many drives have you previously imaged for this custodian: "
read -e DRIVENUMBER  

echo ""
DRIVENUMBER=$((DRIVENUMBER+1))

DATE=`date +%Y%m%d`

IMAGENAME="$CLASTNAME"_"$CFIRSTNAME"-HDD$DRIVENUMBER"-$DATE"

mkdir /mnt/ImageFiles/"$IMAGENAME"  

echo -n "OK, I guess we can start encasing the data now. Go grab another computer, you're wasting time watching me."

echo ""
dc3dd if="$SOURCE" hash=md5 [ ewfacquirestream -w -t
/mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME" -c empty_block -C "$CASENUMBER":"$IMAGENAME" -D "$IMAGENAME" -e "$ILASTNAME", "$IFIRSTNAME" -f encase6 -b 128 -S 2GB ] >
/mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME".log

ewfverify /mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME".E01 2>&1 | tee -a
/mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME".log

echo "" >> /mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME".log

hdparm -i "$SOURCE" 2>&1 | tee -a /mnt/ImageFiles/"$IMAGENAME"/"$IMAGENAME".log
B.1 – DiscoveryAutomator_Lib.EnScript

```en
include "GSI_Basic"

typedef EntryClass[] EntryArray;

class ApplUtilClass {
    static String copyFile(EntryClass fileToCopy, String outputPath) {
        String BasePath = "";
        LocalFileClass lfc = new LocalFileClass();
        EntryFileClass efc = new EntryFileClass();
        NameValueClass fileNames = new NameValueClass();
        String filename = fileToCopy.Name();
        String ext = fileToCopy.Extension();
        ext.ToUpper();
        BasePath = outputPath + "\" + ext + "\"
        BasePath = BasePath.GetFilePath();
        FileHelperClass::CreateFolders(BasePath);
        bool found = false;
        Console.WriteLine("Processing: " + fileToCopy.Name());
        Console.WriteLine("\tAttempting to copy to " + BasePath);
        do {
            found = false;
            if(LocalMachine.PathExists(BasePath + "\" + filename)) { 
                Console.WriteLine("FILE ALREADY EXISTS: " + BasePath + "\" + filename);
                Console.WriteLine("RENAMEING: " + filename);
                //when a duplicate file is found, check if other duplicates exist. if so, update count.
                foreach(NameValueClass nv in fileNames) {
                    if(nv.Name().Compare(filename)==0) {
                        nv.SetValue(int::Convert(nv.Value(),
                        int::DECIMAL) + 1);
                        found = true;
                        int dot = StringHelperClass::FindLastOf(filename, '.');
                        filename = filename.SubString(0, dot) + "[" +
                        nv.Value() + "]" + filename.SubString(dot);
                        break;
                    }
                }
                if (!found) {
                    fileNames.AddNode(filename, 1);
                    int dot = StringHelperClass::FindLastOf(filename, '.');
                    filename = filename.SubString(0, dot) + "]1" +
                    filename.SubString(dot);
                    found = true;
                }
            } else { found = true; }
        } while (!found);
        Console.WriteLine(fileToCopy.FullPath() + "\t" + BasePath + "\"" + filename + "\t" + fileToCopy.Extension() + "\t" + fileToCopy.LogicalSize());
        if (!efc.Open(fileToCopy)) {
            Console.WriteLine("***ERROR***:");
            Console.WriteLine("Unable to Open Entry: " +
            fileToCopy.FullPath());
            efc.Close();
        return fileToCopy.FullPath();
    }
```
if (!lfc.Open(BasePath + "\" + filename, FileClass::WRITE)) {
    Console.WriteLine("***ERROR***: ");
    Console.WriteLine(" Unable to Open File: " + BasePath + "\" +
    filename);
    lfc.Close();
    efc.Close();
    return fileToCopy.FullPath();
}

lfc.WriteBuffer(efc);
lfc.SetTimeStamps(fileToCopy.Created(), fileToCopy.Accessed(),
    fileToCopy/browser()/);
lfc.Close();
efc.Close();
return fileToCopy.FullPath();
}

class DateMatcher{
    DateClass start;
    DateClass stop;

    //Constructor
    DateMatcher(const DateClass & startDate, const DateClass & stopDate): start =
        startDate, stop = stopDate {}

    bool inRange(const DateClass date) {
        if (date.Compare(start) >= 0 && date.Compare(stop) <= 0) {
            return true;
        }
        return false;
    }
}

class ExtensionMatcher{
    StringArray exts;

    //Constructor
    ExtensionMatcher(StringArray extensionList): exts = extensionList {
        forall (String e in exts) {
            e.ToLower();
        }
    }

    bool match(String extension) {
        extension.ToLower();
        forall (String e in exts) {
            if (extension.Compare(e) == 0) {
                return true;
            }
        }
        return false;
    }
}

class SignatureMatcher{
    //Constructor
    SignatureMatcher() {
    }
}
bool match(EntryClass e) {
    return true;
}

class SearchHelper {
  StringArray keywords;
  EntryArray toSearch;
  EntryArray hits;

  // Constructor
  SearchHelper():
    keywords{},
    toSearch{},
    hits{}
  {
  }

  void addKeywords(StringArray words) {
    keywords = words;
  }

  void addKeyword(String word) {
    keywords.Add(word);
  }

  StringArray getKeywords() {
    return keywords;
  }

  EntryArray getSearchFiles() {
    return toSearch;
  }

  void addFile(EntryClass e) {
    toSearch.Add(e);
  }

  EntryArray getHits() {
    return hits;
  }

  void runSearch() {
    SearchClass search();
    for (String keyword in keywords) {
      search.AddKeyword(keyword, KeywordClass::UNICODE | KeywordClass::ANSI);
    }
    search.Create();
    for all (EntryClass e in toSearch) {
      EntryFileClass file();
      int hitNumber = 0;
      if (file.Open(e, 0)) {
        hitNumber = search.Find(file, -1, -1, 0);
        if (hitNumber > 0) {
          hits.Add(e);
        }
      }
    }
  }
}
class Logger {
    String logName;
    String logPath;
    // constructor
    Logger(const String & name, const String & path):
        logName = name,
        logPath = path
    {
        FileHelperClass::CreateFolders(logPath.GetFilePath());
    }
    void log(const String &message) {
        LocalFileClass logFile = new LocalFileClass();
        int openMode = FileClass::APPEND;
        if (!LocalMachine.PathExists(logPath + "\" + logName)) { openMode =
            FileClass::WRITETEXTCRLF; }
        if (!logFile.Open(logPath + "\" + logName, openMode)) {
            Console.WriteLine("Could not open log file: " + logPath + "\" + logName);
        }
        logFile.WriteLine(message);
        logFile.Close();
    }
}
B.2 - DiscoveryAutomator.EnScript

```cpp
#include "GSI_Basic"
#include "Appl_Lib"

class MainClass;

class InputDialogClass: DialogClass {
   MainClass Main;
   PathEditClass ImagePathBox; //Path to hard drive image(ewf)
   CheckBoxClass RecoverDeletedBox; //Checkbox option regarding recovering deleted files
   CheckBoxClass DateFilterBox; //Checkbox option regarding recovering deleted files
   RadioButtonClass DateTypeRadio; //Radio buttons to determine which date field to filter on
   DateEditText StartExportRangeBox; //Starting date range to export
   DateEditText StopExportRangeBox; //Ending date range to export
   ArrayEditText FileExtensionsBox; //Textbox for loose file extensions to export
   ArrayEditText MailExtensionsBox; //Textbox for mail extensions to export
   CheckBoxClass KeywordFilterBox; //Checkbox option regarding recovering deleted files
   ArrayEditText KeywordsBox; //Textbox for keywords to search
   PathEditClass OutputLocationBox; //An output location
   InputDialogClass(DialogClass parent, MainClass main):
      DialogClass(parent, "Appl’s Ultimate Processing Script"),
      Main = main,
      //GUI element constructors
      ImagePathBox(this, "Image Path", START, NEXT, 200, 12, 0, main.imagePath, FILEOPEN),
      RecoverDeletedBox(this, "Recover Deleted Files", START, NEXT, 200, 12, 0, main.recoverDeleted),
      DateFilterBox(this, "Perform date filter", START, NEXT, 200, 12, 0, main.dateFilter),
      DateTypeRadio(this, "Date Filter Field", NEXT, SAME, 200, 12, 0, main.dateType, "Created\tWritten\tAccessed", RadioButtonClass::GROUPBOX),
      StartExportRangeBox(this, "Start Date (MM/DD/YYYY)", START, NEXT, 200, 12, 0, main.startDate),
      StopExportRangeBox(this, "End Date (MM/DD/YYYY)", NEXT, SAME, 200, 12, 0, main.stopDate),
      FileExtensionsBox(this, "File Extensions (one per line)", START, NEXT, 200, 12, 0, main.fileExtensions, 0),
      MailExtensionsBox(this, "Mail Extensions (one per line)", NEXT, SAME, 200, 12, 0, main.mailExtensions, 0),
      KeywordFilterBox(this, "Perform keyword filter", START, NEXT, 200, 12, 0, main.keywordFilter),
      KeywordsBox(this, "Keywords (one per line)", NEXT, SAME, 200, 12, 0, main.keywords, 0),
      OutputLocationBox(this, "Output Path", START, NEXT, 200, 12, 0, main.outputPath, FOLDEROPEN)
   {
   }
}

class MainClass {
   //Variables to grab input from GUI
   String imagePath;
   bool recoverDeleted;
   bool dateFilter;
   int dateType;
   DateClass startDate;
   DateClass stopDate;
}
void mountImage(CaseClass caseClass, const String &imagePath) {
    // Mount the evidence file (must be evidence file), add it's entries to the case and run recover folders
    dbg.log("Attempting to mount evidence file at "+ imagePath);
    uint options = DeviceClass::MOUNTADDENTRIES;
    if (recoverDeleted) {
        options |= DeviceClass::MOUNTRECOVER;
    }
    DeviceClass dev = caseClass.AddEvidenceFile(imagePath, options, null,
        CaseClass::SOURCEEVIDENCE);
    if (dev != null) {
        dbg.log("Mount successful. Should now be running recover folders.");
    } else {
        dbg.log("Mount failed.");
        err.log("Mount failed.");
        SystemClass::Exit("Mount failed");
    }
    dbg.log("Starting verify");
    uint errors;
    if(dev.Verify(errors) == true) {
        dbg.log("Image verified successfully");
        HashClass hash = dev.VerifyHash();
        dbg.log("Verify hash: "+ hash.GetString());
    } else {
        dbg.log("Image verify failed");
        err.log("Image verify failed");
        SystemClass::Exit("Verify failed");
    }
}

//Main method
void Main(CaseClass caseClass) {
    // set output path to case export folder. defined when case is first created
    outputPath = caseClass.ExportFolder();
}
//create three logging objects
log = new Logger("Output.txt", (outputPath+"\"").GetFilePath());
err = new Logger("Error.txt", (outputPath+"\"").GetFilePath());
dbg = new Logger("Debug.txt", (outputPath+"\"").GetFilePath());

dbg.log("Logs created");

SystemClass::ClearConsole();

//Display GUI

dbg.log("Creating GUI");
InputDialogClass diag(null, this);

dbg.log("Getting GUI results");
if (diag.Execute() != SystemClass::OK) {
    err.log("User didn't click OK");
    SystemClass::Exit("Script cancelled by user");
}

//Sanity check values from GUI input

dbg.log("User clicked OK");

dbg.log("Data from GUI:");
dbg.log("ImagePath: "+imagePath);
dbg.log("recoverDeleted: "+recoverDeleted);
dbg.log("dateType: "+dateType);
dbg.log("startDate: "+startDate.GetString(0));
dbg.log("stopDate: "+stopDate.GetString(0));
dbg.log("FileExtensions: ");
foreach (String s in fileExtensions) {
    dbg.log("\t"+s);
}

dbg.log("mailExtensions:");
foreach (String s in mailExtensions) {
    dbg.log("\t"+s);
}

dbg.log("keywords:");
foreach (String s in keywords) {
    dbg.log("\t"+s);
}

dbg.log("outputPath: "+outputPath);
dbg.log("Done checking GUI results");

mountImage(caseClass, imagePath);

dbg.log("Finding files to search");

//Setup matcher classes
DateMatcher dm(startDate, stopDate);
ExtensionMatcher em(fileExtensions);
SignatureMatcher sm();
SearchHelper sh();
sh.addKeywords(keywords);

forall (EntryClass e in caseClass.EntryRoot()) {
    //Ignore folders and overwritten files
    //if (e.IsFolder() || e.IsOverwritten()) {
    if (e.IsFolder() || e.IsOverwritten()) {
        continue;
    }
    DateClass entryDate;
    if (dateType == 0) {
        //Created
        entryDate = e.Created();
    }
} else if (dataType == 1) {
  //Written
  entryDate = e.Written();
} else if (dataType == 2) {
  //Accessed
  entryDate = e.Accessed();
}

dbg.log("Dates:" + " Entry:" + entryDate.GetString(0) + " Start:" + startDate.GetString(0) + " Stop:" + stopDate.GetString(0));
//still need to add check for signature match
if (em.match(e.Extension())) {
  if (dateFilter) {
    if (dmInRange(entryDate)) {
      dbg.log(e.Name() + " in date range & extension matches");
      sh.addFile(e);
    }
  } else {
    dbg.log(e.Name() + " extension matches");
    sh.addFile(e);
  }
}

if (keywordFilter) {
  dbg.log("Starting search");
  sh.runSearch();
  dbg.log("Search complete.");
  searchHits = sh.getHits();
  dbg.log("Search Hits");
  dbg.log("--------");
  forall (EntryClass e in searchHits) {
    dbg.log("t" + e.Name());
  }
} else {
  searchHits = sh.getSearchFiles();
}

dbg.log("Starting File Export");
dbg.log("--------");
forall (EntryClass e in searchHits) {
  log.log(e.Name() + "," + ApplUtilClass::copyFile(e, outputPath));
}

dbg.log("Starting Mail Export");
dbg.log("--------");
ExtensionMatcher me(mailExtensions);
forall (EntryClass e in caseClass.EntryRoot()) {
  if (me.match(e.Extension())) {
    dbg.log("t" + e.Name());
    log.log(e.Name() + "," + ApplUtilClass::copyFile(e, outputPath));
  }
}
Console.WriteLine("Done! Check the case's export folder for error and debug logs.");
C – dtSearchAutomator.java

// dtSearchAutomator.java
// 2012 Joel Bowers

import com.dtsearch.engine.*;
import java.io.File;
import java.util.Scanner;
import java.util.Vector;

public class dtSearchAutomator {
    public static void main(String[] args) {
        Scanner input = null;
        Vector<String> queries = new Vector<String>();

        // check args
        if (args.length != 3) {
            System.out.println("Usage: dtSearchAutomator <stats||export> <path_to_index_file> <path_to_search_strings_file>");
            System.exit(0);
        }

        String task = args[0];
        String indexPath = args[1];
        String filePath = args[2];

        // Get search strings, 1 per line.
        try {
            input = new Scanner(new File(filePath));
            while (input.hasNextLine()) {
                queries.add(input.nextLine());
            }
            input.close();
        } catch (Exception e) {
            System.exit(0);
        } finally {
            input.close();
        }

        // Build search job. Default settings.
        SearchJob job = new SearchJob();
        job.setIndexesToSearch(indexPath);

        // for each query, do a search
        for (String query : queries) {
            job.setRequest(query);
            job.execute();

            if (job.getErrors().getCount() > 0) {
                for (int i = 0; i < job.getErrors().getCount(); i++) {
                    System.out.println(job.getErrors().getMessage(i));
                }
            }

            SearchResults results = new SearchResults();
            results = job.getResults();
            int count = job.getFileCount();

            if (task.equals("stats")) {
                long sizeTally = 0;
                for (int i = 0; i < count; i++) {
                    results.getNthDoc(i);
                }
            }
        }
    }
}
sizeTally += results.getDocSize();
}
System.out.println(query + "": " + count + ", " + sizeTally + 
" bytes");
} else {
for (int i = 0 ; i < count; i++) {
results.getNthDoc(i);

String name = results.getDocName();

System.out.println("robocopy \"testdocs\" \"exportdocs\" " +
name.substring(name.lastIndexOf(\"\") + 1) + "/COPY:DAT");
}
}
}
References


