



**TEMPORAL EVENT ABSTRACTION AND
RECONSTRUCTION**

DISSERTATION

James Okolica

AFIT-ENG-DS-17-D-004

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-DS-17-D-004

TEMPORAL EVENT ABSTRACTION
AND RECONSTRUCTION

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

James Okolica, MS

December 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-DS-17-D-004

TEMPORAL EVENT ABSTRACTION
AND RECONSTRUCTION

James Okolica, MS

Committee Membership:

Dr. G. L. Peterson
Chairman

Dr. R. F. Mills
Member

Dr. M. R. Grimaila
Member

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

Abstract

Digital forensic examiners are being overwhelmed by the increasing demand for their services. Hard drives are growing faster than analysts can review them. While there are many tools to extract data, there are fewer to analyze it and there are none that automatically summarize the raw data into a human understandable narrative. This paper presents a new technique that uses sequence pattern mining (SPM) to generate terms, sequences and rules to summarize digital artifacts into human understandable activities. An existing SPM algorithm, Discontinuous Varied Order Sequence Mining (DVSM), is extended and two new SPM algorithms, Single Object Sequence and Loop Abstraction (SOSLA) and Sequence Mining of Temporal Clusters (SMTC) are created. These algorithms extend SPM by attaching attribute variables to items and by developing a new method for addressing interleaving. First, the algorithms are applied to use case datasets to create sequences. Next, an innovative process automates the consolidation of similar sequences into production rules for discovering abstract user activities in digital forensics data. Finally, application of these algorithms to use cases in a “leave one out” test show a 96% reduction in transactions to review, 100% detection of true positives and a worst case of 0.03% detection of false positives.

AFIT-ENG-DS-17-D-004

To my wonderful life partner and children. Thank you for your love and support.

Table of Contents

	Page
Abstract	iv
Dedication	v
List of Figures	xi
List of Tables	xii
List of Algorithms	xiii
I. Introduction	1
An Example	2
Background	4
1.1 Hypothesis	6
1.2 Definitions	6
1.3 Method	8
Dataset	8
Abstraction	9
1.4 Contribution	10
1.5 Limitations of the Study	11
1.6 Results	11
1.7 Summary	12
II. Background	13
2.1 Sequence Pattern Mining	13
Learning Grammars	15
Sequence Categorization	19
Interleaving	21
Discontinuous Varied-Order Sequential Miner (DVSM)	22
Open Research Questions in Sequence Pattern Mining	23
2.2 Digital Forensics	24
Acquisition	25
Artifacts	26
Timeline Reconstruction	28
2.3 Summary	30
III. Methodology	32
3.1 Research Hypothesis and Evaluation Metrics	32
3.2 Method	33
3.3 Formal Description	33

	Page
Attaching Variables to SPM algorithms.....	36
3.4 Constraint Generation	40
3.5 Term, Sequence and Production Rule Generation	43
Single Object Sequence and Loop Abstraction (SOSLA)	44
Discontinuous Varied Order Sequence Mining with Variables (DVSM-V)	45
Sequence Mining of Temporal Clusters (SMTC)	47
3.6 Dataset Abstraction	52
3.7 Rule Consolidation	54
3.8 Summary	59
IV. Results and Analysis	60
4.1 The Data	60
Experiment	61
4.2 Database and Algorithm Design and Experiment Setup	62
Data Extraction	62
Database	63
Algorithms	64
4.3 Data Abstraction	65
Term and Sequence Generation	65
Production Rule Generation	67
4.4 Results	67
4.5 Summary	69
V. Conclusions	75
5.1 Hypothesis	75
5.2 Algorithms	76
5.3 Experimental Methodology	77
5.4 Results	78
5.5 Discussion	78
5.6 Next Steps	79
5.7 Impact	79
Appendix A. Scenarios	80
1.1 Use Case 1 - Microsoft (MS) Word in Windows 10	80
1.2 Use Case 2 - Microsoft (MS) PowerPoint in Windows 10	83
1.3 Use Case 3 - Microsoft (MS) Excel in Windows 10	86
1.4 Use Case 4 - Google Chrome in Windows 10	89
1.5 Use Case 5 - Mozilla Firefox in Windows 10	91

	Page
Appendix B. Production Rules for Starting Applications	93
2.1 Use Case 1 - Microsoft (MS) Word in Windows 10	93
2.2 Use Case 2 - Microsoft (MS) PowerPoint in Windows 10	102
2.3 Use Case 3 - Microsoft (MS) Excel in Windows 10	114
2.4 Use Case 4 - Google Chrome in Windows 10	132
2.5 Use Case 5 - Mozilla Firefox in Windows 10	153
Appendix C. Parameter File	172
Appendix D. Algorithms	176
4.1 Single Object String and Loop Abstraction (SOSLA)	176
Header	176
C++	177
4.2 Discontinuous Varied Order Sequence Mining with Templates (DVSMT)	181
Header	181
C++	182
4.3 Sequence Mining of Temporal Clusters (SMTC)	191
Header	191
C++	192
4.4 String Abstraction (AbstractStrings)	203
Header	203
C++	203
4.5 Loop Abstraction (AbstractLoops)	215
Header	215
C++	215
Appendix E. Scripts	222
5.1 Use Case Dataset	222
5.2 Real World Dataset	223
5.3 RegRipper Scripts	229
Event Messages	229
Date and Time for all Registry Keys	232
Primary Registry Extraction Script	236
Appendix F. Terms and Strings	273
6.1 Generics	273
Header	273
C++	274
6.2 GenericsXref	277
Header	277
6.3 InstanceGenericsXref	279

	Page
Header	279
6.4 Patterns	281
Header	281
C++	283
6.5 Terms	316
Header	316
C++	318
6.6 Strings	326
Header	326
C++	328
6.7 ActivityBag	357
Header	357
C++	358
Appendix G. Events	370
7.1 Timeline	370
Header	370
C++	371
7.2 TimelineEntry	385
Header	385
7.3 TermList	389
Header	389
7.4 ItemInstance	391
Header	391
7.5 Print	398
Header	398
C++	399
7.6 Search	414
Header	414
Header	416
Appendix H. Miscellaneous Source Code	434
8.1 TearCPPCPP	434
Header	434
C++	436
8.2 Parameters	442
Header	442
C++	444
8.3 BuildDB	453
Header	453
C++	457
8.4 ItemUtilities	514
Header	514

	Page
C++	515
8.5 Utilities	525
Header	525
8.6 queues	558
Header	558
8.7 stacks	561
Header	561
8.8 lists	563
Header	563
Bibliography	571

List of Figures

Figure		Page
1.	Floating Point Example.	16
2.	Greedy Node Merging Algorithm Example.	18
3.	Attaching Variables to SPM.	38
4.	Traceability.	39
5.	Abstraction Methodology.	44

List of Tables

Table	Page
1. Acquisition Tools.	24
2. RegRipper Plugins relevant to Temporal Artifacts.	27
3. Extracted Data Files.	61
4. Use Cases.	62
5. Database Event Tables.....	63
6. Database Production Rules.....	63
7. Database Traceability.....	63
8. Use Case Dataset Compression.....	65
9. Use Case Dataset Compression.....	66
10. Representative Terms and Sequences.....	70
11. Example Sequences for Starting Microsoft Word.....	71
12. Representative Production Rules.	71
13. Example of Production Rules Generating a Sequence.....	72
14. Production Rule Testing for SMTC.	73
15. Production Rule Testing for DVSMTC.	74

List of Algorithms

1	String-to-String Correction Problem.	20
2	SPM - Variable Attachment (part 1).....	40
3	SPM - Variable Attachment (part 2).	41
4	Single Object Sequence and Loop Abstraction (SOSLA).....	46
5	Discontinuous Varied order Sequential Mining with Variables (DVSM-V).	48
6	DVSM-V - Resolve Variants.	49
7	DVSM-V - Keep Interesting.	50
8	Sequence Mining of Temporal Clusters (SMTTC).....	53
9	SMTTC -Keep Interesting.	54
10	Sequence Abstraction.....	55
11	Loop Abstraction.	56
12	Sequence Consolidation.	57
13	Build Graph.....	58

TEMPORAL EVENT ABSTRACTION AND RECONSTRUCTION

I. Introduction

One of the biggest open issues in Digital Forensics is the continuing increase in numbers of cases compounded with the increase in digital media storage space. Digital Forensic Laboratories cannot keep up [63, 82]. To put this in perspective, a 4 terabyte hard drive costs \$130 [107] and can store 80 million pictures or 2,000 hours of video [82]. Assuming an examiner watched video manually, that equates to 250 8-hour days or 50 40-hour workweeks.

While first generation tools have evolved to scale to the larger storage space, there are still more *digital artifacts*, i.e., digital traces of evidentiary interest, for examiners to wade through. In the face of so much data, there have been several suggestions to limit what is collected [81, 83, 8, 86]. While this may be necessary in the short term in order to provide timely results, it runs the risk of missing digital artifacts that may be relevant. One possible solution is digital forensic triage where all data is collected but examination of the data is prioritized [78, 94]. Quick and Choo [82] also make a distinction between digital forensic data as evidence versus as intelligence. Evidence has a legal connotation whereas intelligence is “knowledge designed for action.” A second issue, related to the first, is the need to abstract digital artifacts into human understandable activities, making it possible to handle more data quicker. Previous research on abstraction of digital forensic data into human understand activities either involves sensors in place prior to incidents [54] or hand generated rules by subject matter experts [51, 35].

An Example.

The historian Lawrence Stone [97] defines a *narrative* as “organized chronologically; it is focused on a single coherent story; it is descriptive rather than analytical; it is concerned with people not abstract circumstances; and it deals with the particular and specific rather than the collective and statistical.”

Digital forensics seeks to create two narratives [80]. The first is a *technical narrative* that describes the provenance on the data: when it was created, how it was created, where it was created, etc. The second is an *investigative narrative* that places the data recovered within the larger legal narrative. The technical narrative is vital but only provides the skeleton around which the investigative narrative is built. The goal is to provide “actionable intelligence, investigative leads, testimony and/or probative evidence.” [80]

The following example provides an illustration of the problem. Consider a case where someone dies unexpectedly. The police arrive on the scene and initial indications are a suicide. The victim’s computer is on and there is a suicide note open on the screen. Did the suspect write the note? If so, when? If not, who did?

In our example, the police contact a digital forensic examiner. The examiner may first perform an incident response. They attach a USB drive and create a copy of volatile memory. The examiner then seizes the hard drive. As soon as he returns to the lab with the drive, the examiner images the drive and makes a copy of the volatile memory and hard drive images so that the originals are left untouched. Investigating the images will either reject or fail to reject the hypothesis that the suicide note is genuine. The examiner begins by testing the hypothesis that there was malware installed on the suspect computer. He opens a tool that parses the volatile memory image to see what processes were active. There are 64 active processes and all but 2 are normal system processes, all started by the system with executables in the system

directory. The remaining two processes are Microsoft Word and Internet Explorer. Internet Explorer was started at 6:07:21 pm EST on the day of the victim's death. Microsoft Word was started about two hours later, at 8:14:11 pm EST, approximately at the same time as the coroner's initial projected time of death. There is no evidence of malware and the information gleaned from the active processes supports the initial hypothesis that the suicide note was written by the victim.

The next hypothesis is that someone other than the victim logged into the computer. The number of mis-typed passwords might support this hypothesis. The examiner mounts the hard drive image and retrieves the Microsoft Windows Registry files along with the Windows event logs. A tool that examines the Windows event logs shows a large amount of data, so the examiner limits the event to the 24 hours preceding the victim's death. There are still a large number of events, but most of them are related to a software update that was applied the night before the victim's death. The examiner makes a note to check on the software update later to see if it was legitimate. Continuing to examine the event log, he finds that she logged into her computer at 6:04:32 pm EST on the day of her death. There were no instances of a mis-typed password at that time.

Next, the examiner examines the Windows Registry, which is a mini file-system that Microsoft created for configuration information for system and user applications. While there are conventions for the root folders, sub-folders may be created by users and system applications as desired. Attempting to review all of the registry entries is unrealistic so he limits it to those registry keys that he has previously found to be useful. By looking at recently accessed files, the examiner can check if the suicide note was created at the time of death or at some point earlier. One of the registry keys of interest is the recently used files by Microsoft Word. Looking at that key, the examiner finds that the last file used was %USERPROFILE%/Downloads/Suicide.doc.

Because the file is in the downloaded file directory, a hypothesis is that the file was downloaded rather than created locally. To test this, the examiner returns to the hard drive image and retrieves the Internet Explorer log files. The last few entries support the hypothesis that `suicide.doc` was indeed downloaded. Further supporting it is the log of the last websites visited that shows the last website was Yahoo Mail but wasn't the victim's email address.

At this point, the examiner contacts the police assigned to the case and says something like, "I've reviewed the computer. She logged into the computer at around 6:00 and opened Internet Explorer. Two hours later, someone accessed `joe.smith@yahoo.com`, downloaded a file titled `suicide.doc` and opened it in Microsoft Word. This is all preliminary and I'll have more details later." In a couple of short sentences, the examiner has taken all of the data examined and synthesized a narrative. Automating this process is the problem that the current research addresses.

Background.

The Oxford dictionary defines *abstraction* as "the quality of dealing with ideas rather than events" [30]. In the context of this research, it means to replace the digital artifacts retrieved from *digital media*, the digital content encoded for storage, presentation or processing on a digital electronic device, with human understandable ideas.

For the purpose of this research, the events that occur on digital devices are described by a start time, an end time, and a digital artifact that provides a record of the action. The digital artifact includes an *actor* that performed the action (either the system or a user), the *action* performed, the *application* (if any) from which the action occurred, and a *digital object* that the action is performed on. *Digital objects* are organized collections of bits that provide content to either a digital device or the

user of a digital device. Digital objects include files, registry keys, event logs, as well as other content.

Consider digital objects in the Windows NT Master File Table (MFT). Each file stored on digital media has additional digital artifacts stored with it that include the time the file was created, the last time it was modified, the last time it was accessed, and the last time its meta information was changed. Each of these times corresponds to a separate action, respectively the file being created, the file being modified, the file being accessed, and the file's meta information being changed. Consider the unique case where all four actions occurred at the same time (e.g., the file was created but nothing has been done with it since). Summarizing those four actions into an "A file was created at ___" idea provides the same amount of high level information to an examiner but with a 75% reduction in the amount of data that needs to be read.

Beyond the MFT, there is a large amount of temporal information available on a digital device. Modern file systems record the last time a file was accessed and changed as well as when it was created (e.g., Microsoft Windows NT Master File Table). Office automation applications keep track of the most recently used files (e.g., Microsoft Windows Registry) to improve user productivity. Web browsers keep track of the websites visited and the files downloaded (e.g., web logs). And the operating system keeps track of important system events like user logins (e.g., Microsoft Windows Event Log) and active processes (e.g. volatile memory image).

Discovering commonalities between multiple occurrences of the same general events and then using these discoveries to develop transformation between digital artifacts and human understandable activities enables the overwhelming number of digital artifacts to be summarized into a few sentences that describe user activities.

1.1 Hypothesis

This research develops and presents algorithms that take transactional data from digital media, create sequences from it, and then use those sequences to develop production rules for discovering abstract user activities. The hypothesis is that “sequence pattern mining can be used to automate abstracting digital forensic artifacts into human understandable activities.”

1.2 Definitions

The following definitions are used throughout the document for the associated terms. They are placed here for clarity and to provide a single reference point. The first time each term is used in the document, it is italicized.

- A *digital artifact* is a digital trace found on digital media that is of evidentiary interest. Digital artifacts serve several purposes, including as a record of a change to the digital media and as a record of the digital objects on the digital media.
- A *digital object* is an organized collection of bits that provide content to either a digital device or a user of a digital device. Digital objects include several types of content, all of which must be predefined.
- An *action* is the process of doing something on a digital media. Digital artifacts provide a record of actions.
- An *actor* is either the operating system or a user that performs actions.
- An *application* is a software program within which an actor perform actions on the digital media.

- An *item* is a unit in a sequence.
- A *sequence* is an ordered set of items.
- An *attribute* is a quality or feature of an item.
- A *variable* is a symbol, associated with an attribute, that may assume a given value subject to specified constraints.
- A *term* is an item with one or more attributes, each with a variable assigned to it. For this research, a simple term consists of an actor, an action, an application (which may be null), and a variable which represents zero or more digital objects. A term may also be an abstraction (defined below).
- A *dataset* is a sequence of items. For this research, an item is either a digital artifact or a term.
- An *abstraction* is the replacement of a sequence of items with a single term.
- A *loop* is an abstraction where all of the terms in the sequence being replaced are the same and differ only in their attribute values.
- A *single-object sequence* is an abstraction where the variable of each of the terms in the sequence being replaced all have the same value.
- A *multi-object sequence* is any abstraction that is not a loop or single-object sequence.
- *Production Rules* are relations between digital artifacts, terms, and sequences
- *Conditional variables* are variables that may only take on a subset of values. For instance, *a*, *b*, and *c* may only take on rational values between \$0.01 and \$100.00. Any time these variables are associated with a term the value of the attributes

that the variables are assigned to will be within the constrained space. Within the term, the relationship between the variable, attribute and value is called a *cross reference*.

- An *instantiation* of a term is a specific sequence of the item(s) that make up the term in the dataset. Associated with the instantiation are its variable values.
- The *workspace* is a sequence in a dataset that is currently being examined by an SPM.
- A *human understandable activity* is an action or series of actions expressed in terms a non-technical person would understand. For instance, “starting Microsoft Word” rather than listing all of the files accessed when that occurs.
- *Interleaving* is a sequence created by alternating, in random order, the terms of two smaller sequences

1.3 Method

The human understandable activities are developed through four steps. First, digital media is acquired. Next digital artifacts are extracted. Third, the digital artifacts are used to create terms, sequences, and production rules. Finally, these terms, sequences and production rules are applied to new datasets to discover human understandable activities from and provide them contextually to the examiner at whatever level of detail is desired.

Dataset.

The dataset used in this research is constructed using scenarios for user activities. A user performs each step in the scenario and records the time. Later the start and stop times of each scenario are used to transform mined sequences of digital artifacts

into production rules that describe human understandable activities. Relevant digital artifacts are extracted including file times from Microsoft Windows NTFS Master File Table, event logs from the system log files, system and application settings from the Microsoft Windows registry, and browsing activity from web browsers.

After dataset acquisition, the digital artifacts from the extracted files are coded into a sequence of items where item is composed of a start time, stop time and the digital artifact. The digital artifacts consist of an actor, action, software application (possibly null), and a digital object. The actors are either user login names or the system name. Actions include create a file, access a file, modify a file, change meta information on a file, download a file, visit a webpage, connect a device, connect to a network, modify a registry key, as well as many that are defined based on the dataset.

Abstraction.

Once the digital artifacts are imported, the next step and the primary purpose of the research is to abstract them into human understandable activities. The dataset is mined using SPM algorithms and then production rules are generated from the resulting sequences present during the start and stop times of each scenario of a use case.

Once the terms, sequences, and production rules are generated, they are applied to the dataset to create human understandable activities. To summarize, this research takes digital artifacts, uses those artifacts to create terms, sequences, and production rules and then applies them to a dataset to create human understandable activities. The output is the terms, sequences, production rules and the abstraction of the digital artifacts into human understandable activities.

1.4 Contribution

While there is extensive research in sequence pattern mining (SPM), there is little research applying it to digital forensics. Digital forensics adds the additional complexity that items have digital objects associated with them (e.g., opening file `123.txt` in a word processor). Rather than create a separate item for every different digital object, the SPM algorithms developed in this research assign variables to items where the variable's value is the digital object's name. This enables a single item to tie a specific digital object (e.g., file `123.txt`) to a specific instantiation. This research creates one new pre-processing SPM, extends an existing SPM to include the variables, and creates a new SPM that resolves interleaving issues in a new way. *Interleaving* occurs when two sequences occur concurrently. The result is a larger sequence created by alternating, in random order, the terms of two smaller sequences. The issue is that during SPM, these two sequences need to be separated so that the relevant individual sequences can be mined.

The first of the new algorithms, Single Object Sequence and Loop Abstraction (SOSLA), is a greedy algorithm that constructs sequences of terms where the actor, action, and application may vary but the object does not. When the single object restriction is removed from SOSLA and a small sequence of time is specified, SOSLA-Multi Object (SOSLA-M) creates a single sequence that completely describes everything that occurs during the time interval (e.g., an individual editing a spreadsheet in Microsoft Excel).

The existing sequence pattern mining algorithm, Discontinuous Varied Order Sequential Mining (DVSM), begins by constructing all 2-item sequences that occur in the dataset and then keeping those of interest. It then uses these interesting 2-item sequences to create 3-item sequences. It does so by adding two 3-item sequences, one that includes the 2-item sequence and the item that immediately preceded it and one

that includes the 2-item sequence and the item that immediately followed it. This process of extending the sequences and then pruning the result continues either for a fixed number of iterations or until no more sequences are generated. It then clusters similar sequences together, forming a main sequence and variations on it. This research extends DVSM by adding variables (DVSM-V) that allow digital objects to be attached to the items.

Finally, the second new algorithm, Sequential Mining of Temporal Clusters (SMTC), provides a new approach for addressing interleaving. SMTC begins by grouping together all items that occur within a small period of time. It then creates all possible subsets of these items. It then mines these subset to find those of interest. Only after it has reduced the subsets to a smaller number does it create a representative sequence for each subset.

1.5 Limitations of the Study

This research has several limitations. First, no effort is made to obfuscate any user activities. Second, the limitations of the hardware used restrict the total number of terms in the dataset and the number of items in the individual generated sequences. Finally, further testing would likely result in better parameter selection for the thresholds of what constitutes similarity between sequences and what is sufficient to consider a sequence interesting enough to keep.

1.6 Results

The dataset is composed of five use cases and between five and eight scenarios for each use case. The algorithms are applied to the dataset to create terms and sequences. For each use case, N experiments are performed. For each experiment, one scenario is left out and production rules are generated on the other $N - 1$ scenarios.

Then, the production rules are tested on the previously unseen data. The term and sequence creation reduces the events to be reviewed by 96%. The “leave one out” testing results in 100% detection of the true positives and no more than a 0.03% detection of false positives.

1.7 Summary

Chapter 2 begins with a more detailed review of sequence pattern matching (SPM) research and then proceeds to describe the challenges facing digital forensic examiners and the applicability of SPM research. Chapter 3 begins with the research hypothesis and metrics and then presents a formal representation of the domain. It then describes the overall sequence and rule generation and dataset abstraction methodology before providing a high level overview of the principal algorithms and concluding with a discussion of the dataset generation. Chapter 4 details some of the implementation decisions made during the development of the algorithms before proceeding to the results of the experiment. Finally, Chapter 5 concludes with lessons learned and a discussion of planned future work.

II. Background

The goal of this research is to answer “can sequence pattern mining be used to automate abstracting digital forensic artifacts into human understandable activities?” This chapter discusses current research in each of these areas, i.e., sequence pattern mining, digital forensics, and timeline reconstruction.

There are several potential areas of related work that are not discussed. The research is focused on digital devices where the user’s activity is of interest as it relates to a larger, non-cyber investigation. Therefore, cyber attacks, incident response and anti-forensic techniques are neither covered in this related work nor addressed in the methodology.

2.1 Sequence Pattern Mining

Aggrawal and Srikant [1] describe SPM as “ *Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns with a user-specified minimum support, where the support of a pattern is the number of data-sequences that contain the pattern.*”

SPM algorithms fall into three main categories [72]. The first, Apriori-Based Algorithms, extend the original algorithms developed by Agrawal [2]. The second, Pattern Growth Algorithms, addresses Apriori’s need for creating a large number of candidates during multiple passes through the dataset by creating a frequent pattern tree and then dividing the tree into a set of projected databases. Finally, the third type of algorithm, Temporal Sequence Mining Algorithms, mine datasets that are episodic in nature [66] where an episode is “a collection of events that occur relatively close to each other in a given partial order.” Observe that while all three types

of algorithms are appropriate to temporal data, i.e., data that is sequenced by a timestamp, the temporal sequence mining algorithms address a partial ordering, e.g., where an item may occur over a span of time that overlaps a span of time when a second and third item occur.

The general process the Apriori techniques use is to make multiple passes through a database of item. The first pass creates sequences of length 1 and stores the most frequent ones. The second pass extends the frequent length 1 patterns by one to length 2 patterns and stores the most frequent length 2 patterns. The process repeats until no new patterns are added. Over the years, there have been several methods applied to improve the speed of these algorithms [115, 100]. However, the improvements have been evolutionary rather than revolutionary. For recent surveys of SPM techniques see [64, 72]

In general, SPM algorithms either ignore item attributes or use them to define constraints a priori. For instance, there are algorithms [56, 14] that might search for computer and printer purchases where the price of the computer is more than \$1,000 and the price of the printer is less than \$200. While constraints work if mining for specific types of constraints, they don't work when part of the mining is discovering what those constraints may be (e.g., what are the best price breaks for computers and printers). One option is to create duplicate items for each item/constraint combination as they are mined. However, this can become prohibitive. An alternative is keeping a single item with an attribute variable and embedding the value within the item.

SPM Algorithms that do handle item attributes through the use of variables include SeqLog [27] and MiTempP [59]. Lee and Raedt [27] created SeqLog as a logical language that handles sequences of logical atoms with variables attached to them. Lattner et. al.[59] extend this work with MiTempP which handles temporal data.

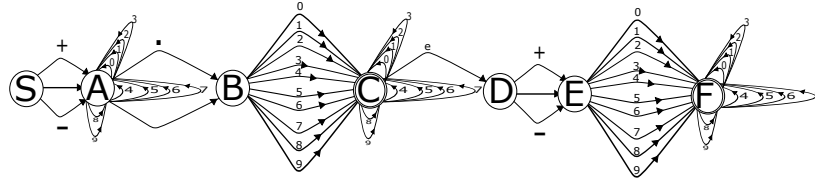
Both SeqLog (and its associated algorithm, MineSeqLog) and MiTempP are designed to work in domains where there are a small number of items that are defined a priori and where interleaving is not an issue. To the best of our knowledge, no SPM approach handles large numbers of items, item discovery during mining, interleaving, and variables. Observe that there are algorithms that mine for sequences subject to constraints [56, 14]. These algorithms constrain the sequences they find based on item attributes. However, this differs from the current work in that these Constraint Frequent Set Queries search for sequences where the constraints are defined a priori. The current work searches for sequences where the constraints are defined as part of the mining process.

Learning Grammars.

A related perspective comes from the natural language processing field. Rather than looking at activities as discreet items or sequences to be categorized, they can be looked at as a continuous stream of phrases and thoughts. For instance, creating a document can be thought of as the sentence, “**create, modify, access, and change**”. These same words can be combined with other words and in other orders to form other sentences. A finite state automaton (FSA) [46], M , over an alphabet, σ is a system $(K, \sigma, \delta, q_0, F)$, where K is a finite, non-empty set of states, σ is a finite input alphabet, δ is a set mapping $K \times \sigma$ into K , $q_0 \in K$ is the initial state, and $F \subseteq K$ is the set of final states. This same definition is used for regular grammars with some minor changes. A regular grammar is defined by $G = \{V, \Sigma, P, S\}$ where V is a finite set of variables (similar to the K states), Σ is the alphabet (like σ), and S is the initial variable (like q_0) [98]. There are no final states since with regular grammars we are concerned more with the journey (i.e., the path) than the destination. P , similar to δ , is a set of production rules that are of the form $B \rightarrow a$, $B \rightarrow aC$, and $B \rightarrow \epsilon$

where $B, C \notin F$, $a \in F$ and ϵ is the empty set. A sentence, x is accepted by a finite automaton, M , if $\delta(q_0, x) = p$ for some p in F . Any sentence that is accepted by a finite automaton is said to be a regular expression. The set of all regular expressions accepted by a Finite State Automata is a regular grammar. As shown in Figure 1 FSAs are generally represented visually (where S is the initial state and double circles are final states) while regular grammars are represented as a set of rules. Consider the production rules take $B \rightarrow a$, $B \rightarrow aB$, and $B \rightarrow \epsilon$ where $B \notin F$, $a \in F$ and ϵ is the empty set. Observe that these rules generate the set $\{\epsilon, a, aa, aaa, \dots\}$. If instead the set $\{a, aa, aaa, \dots\}$ is desired, the third rule (i.e., $B \rightarrow \epsilon$) is removed. Finally, if the second rule (i.e., $B \rightarrow aB$) is removed, the resulting set is simply $\{a\}$. A shorthand way of writing these sets of rules are, respectively, $B \rightarrow a^*$, $B \rightarrow a^+$, and $B \rightarrow a$.

Finite State Automata



Regular Grammar

$S \rightarrow +A$	$A \rightarrow 0A$	$B \rightarrow 0C$	$C \rightarrow 0C$	$D \rightarrow +E$	$E \rightarrow 0F$	$F \rightarrow 0F$
$S \rightarrow -A$	$A \rightarrow 1A$	$B \rightarrow 1C$	$C \rightarrow 1C$	$D \rightarrow -E$	$E \rightarrow 1F$	$F \rightarrow 1F$
$S \rightarrow A$	$A \rightarrow 2A$	$B \rightarrow 2C$	$C \rightarrow 2C$	$D \rightarrow E$	$E \rightarrow 2F$	$F \rightarrow 2F$
	$A \rightarrow 3A$	$B \rightarrow 3C$	$C \rightarrow 3C$		$E \rightarrow 3F$	$F \rightarrow 3F$
	$A \rightarrow 4A$	$B \rightarrow 4C$	$C \rightarrow 4C$		$E \rightarrow 4F$	$F \rightarrow 4F$
	$A \rightarrow 5A$	$B \rightarrow 5C$	$C \rightarrow 5C$		$E \rightarrow 5F$	$F \rightarrow 5F$
	$A \rightarrow 6A$	$B \rightarrow 6C$	$C \rightarrow 6C$		$E \rightarrow 6F$	$F \rightarrow 6F$
	$A \rightarrow 7A$	$B \rightarrow 7C$	$C \rightarrow 7C$		$E \rightarrow 7F$	$F \rightarrow 7F$
	$A \rightarrow 8A$	$B \rightarrow 8C$	$C \rightarrow 8C$		$E \rightarrow 8F$	$F \rightarrow 8F$
	$A \rightarrow 9A$	$B \rightarrow 9C$	$C \rightarrow 9C$		$E \rightarrow 9F$	$F \rightarrow 9F$
	$A \rightarrow .B$		$C \rightarrow eD$			$F \rightarrow \epsilon$
	$A \rightarrow B$		$C \rightarrow \epsilon$			

Figure 1. Floating Point Example.

There has been much research done in inferring grammatical expressions from

positive and negative examples. The earliest work is not specifically related to regular grammars; rather, it is applied to creating sequential circuits. Huffman [49] was the first to produce a general algorithm for creating sequential circuits from a given set of acceptable flows. This early work was limited to an alphabet of two characters (i.e., 0 and 1), but it introduced some general techniques for creating finite state machines. Observe that since any character can be translated into a sequence of 0s and 1s, any finite state machine with an alphabet of more than two characters can be transformed into a (not necessarily minimal) finite state machine of two characters where the additional states and transitions handle the transformation of the more complex characters to 0 and 1. The issue with the early work is that it required enough positive examples to fully define the machine. For instance, as shown in Figure 2, Trakhenbrot and Barzdine [102] provide a simple greedy algorithm that finds the smallest deterministic finite automata (DFA) given all of the possible sequences it produces. It begins by creating a tree that starts from a common root and creates every sequence by creating a new node from the root whenever the suffix of a new sequence differs from a previously created one. It then minimizes the tree by merging nodes whenever every lower node of one candidate is contained within a sub-tree of the other candidate. The authors show that the algorithm creates the tree in $m \cdot n^2$ time where m is the size of the initial tree and n is the size of the final tree. Bierman [10] introduced the concept of a non-deterministic machine that produces a more or less precise machine depending on the amount of input. Lang [57] showed that, in general, a variation of Trakhenbrot’s algorithm can find the smallest DFA from sparse positive and negative training examples (3% of the total) if the target concepts and training sets are drawn from uniform distributions. Lang, et al. [58] show that by changing the search for merges from breadth first to an ordering of merges by greatest support the algorithm’s performance improves. Finally, Angluin [5] provides additional theoretical

general results for cases with only positive examples. For a bibliographical study of grammatical inference see Higuera [28, 29].

Regular Expressions

ABCDEF
ABCCEF
ACGDEF
CEF

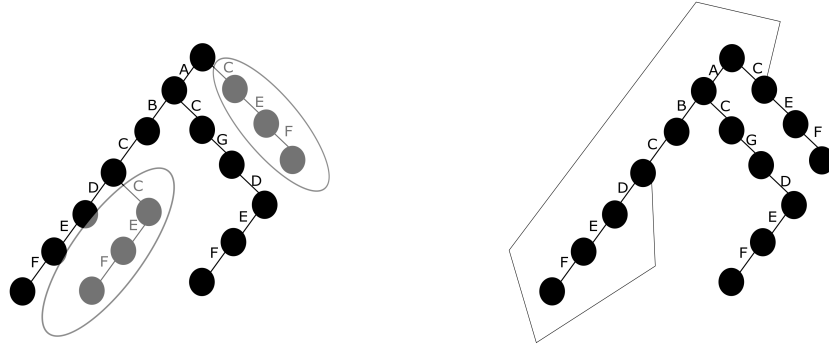
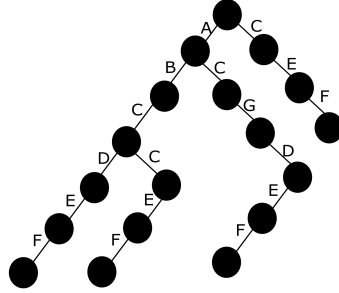


Figure 2. Greedy Node Merging Algorithm Example.

While regular grammars and finite state automata are sufficient in many cases, they are limited. When more expressiveness is required, context-free grammars are used. Context-free grammars (CFG) have most of the same formalism as regular grammars [46]. Both have an alphabet, a start state, and a set of rules. Unlike regular grammars, CFGs do not have a specific set of final states. Instead, they have an explicit set of variables, i.e., items that eventually will be filled by elements in the alphabet. The biggest difference between CFGs and regular grammars is the restriction of the rules. In both, only a single variable may be present on the left side. However, while in regular grammars, the right hand of each rule is restricted to at most one variable and one terminal (and the variable must always be either on

the right or left hand side), no such restriction exists for CFGs. With this additional power, CFGs are able to fully describe natural languages such as English. Early work on inferring CFGs from data include [92, 96]. [28, 29] provides a more recent overview of techniques for discovering CFGs.

Sequence Categorization.

Once sequences have been mined from a dataset, the next step is to group similar sequences together. When there are no training samples to indicate how the sequences should be labeled, the classification needs to be unsupervised (i.e., without training data). Clustering is a popular method of unsupervised classification. In unsupervised learning, the number of categories is not known ahead of time and there are no examples to train from. Instead, the goal is to separate a collection of items into “natural” categories and discover the hidden structure. Patterns in the same cluster should be similar to each other and patterns in different clusters should not [111].

Over the past fifty years, there has been a lot of research in clustering [6, 7, 113, 50, 93]. The most common clustering algorithm is k-means. In its simplest form, it begins by assuming a predefined number of clusters, k . It randomly chooses k items as the centroids for the clusters. Then it examines each remaining item one at a time, placing it in the cluster whose mean is closest to it and then re-calculating the mean of that cluster. The algorithm continues to look at each item multiple times until some threshold is met. There are many variations on this algorithm. Some variants run multiple passes trying to determine the optimal number of clusters. Others adjust not only the mean of the selected cluster, but also the mean of some of the surrounding clusters. Still others allow items to be members of multiple clusters. However, what all of these algorithms have in common is the need to compute the *distance* or *similarity* between two points.

Determining the distance between two items is at the center of most clustering algorithms. One common method used for sequences, the Levenshtein edit distance [60], calculates the amount of effort required to change one sequence into another. There are three actions, add, delete, and substitute where each action can have a different cost (and potentially a different cost for each term). For instance, assuming the cost for each operation is 1, the simple edit distance between *cake* and *castle* is 4 (substitute – case, substitute – cast, add – castl, add – castle). Navarro [73] provides an overview of the different algorithms for efficiently computing edit distance. Algorithm 1 shows one of the earliest algorithms developed by Wagner for computing the distance D between two sequences [105].

Algorithm 1: String-to-String Correction Problem.

```

 $D[0, 0] = 0;$ 
for  $i = 1 : |A|$  do
   $D[i, 0] = D[i - 1, 0] + \text{DeleteCost}(A(i));$ 
for  $j = 1 : |B|$  do
   $D[0, j] = D[0, j - 1] + \text{InsertCost}(B(j));$ 
for  $i = 1 : |A|$  do
  for  $j = 1 : |B|$  do
     $m_1 = D[i - 1, j - 1] + \text{SubstituteCost}(A(i), B(j));$ 
     $m_2 = D[i - 1, j] + \text{DeleteCost}(A(i));$ 
     $m_3 = D[i, j - 1] + \text{InsertCost}(B(j));$ 
     $D[i, j] = \min(m_1, m_2, m_3);$ 

```

A second method for computing similarity of sequences is the sequence clustering algorithm CLUSEQ [114]. Yang begins by computing a imprecise probability suffix tree (IPST) modeled from Bejerano and Yona’s probability suffix tree [9]. The tree establishes an upper and lower bound on the probability of the next character in a sequence given the presence of the previous sequence of characters. Then the probability of a candidate sequence is calculated based on the IPST for each cluster and the sequence is placed in the cluster with the highest probability. Note that this

assumes that the conditional probability distribution (CPD) is the same for all the transitions in a sequence. Lang also uses a threshold to control cluster quantity. If a sequence fails to surpass the threshold for any cluster, it is considered an outlier. Once more than a certain percentage of sequences are considered outliers, one is chosen to create a new cluster. Finally, Lang also introduces a concept of node convergence. If the lower and upper bounds of a node in an IPST differ by more than a specific amount, the node is considered to not have converged. This may mean that the IPST contains multiple CPDs and needs to be split. The entire algorithm is called CLUSEQ and has been applied to many pattern sequence problems. Observe that inherent in the IPST is that each node has τ children where τ is the number of terms in the alphabet. Recently, Xiong, et al. [110] has taken the CLUSEQ algorithm and added to it. First, similarity is now based on patterns from the candidate pattern in the cluster *and* patterns in the cluster that are not in the candidate pattern. Second multiple correspondence analysis (MCA) is used to choose the initial seeds for the clusters, determine similarity, and split the clusters.

A second unsupervised classification algorithm is Latent Dirichlet Allocation (LDA) [11]. Although it was originally designed for text mining, it has been applied to several other domains, including most recently activity discovery and recognition [24]. The biggest advantage of LDA is that unlike other unsupervised learning algorithms like clustering, LDA doesn't need a distance measure. Instead it uses mixture distributions to group similar sequences.

Interleaving.

Much of the research in SPM assumes that the data is strictly sequential and, excepting noise, is uninterrupted. However, in many cases these are not reasonable assumptions. For instance, consider activities of daily living (ADL). When a person

fixes breakfast in the morning, they will often start something cooking on the stove while setting the table and later may have their meal interrupted to answer the telephone. In both cases, activities are *interleaved* together. That is, the individual activities of two sequences alternate. Recently, there has been a significant amount of research into interleaving in the ADL domain including skip-chain conditional random fields [47], genetic algorithms [91], interleaved hidden Markov models [71], Markov logic networks [45], emerging patterns [39] and finite automata [3]. One concern is that the dataset common to several of these approaches is [79]. The dataset has a small number (60) of different activities and it appears that there are under 10 activities in a given activity sequence. It is unclear whether the approaches scale to a larger alphabet or to longer sequences.

Discontinuous Varied-Order Sequential Miner (DVSM).

While most of the research has focused on either discovering sequences or categorizing them, there has been some that combines the two. Rashidi and Cook [84, 26] have created the Discontinuous Varied-Order Sequential Miner (DVSM) algorithm to discover frequent sequences and then cluster sequences that likely belong together. DVSM begins by scanning the collection of activities and creating sequences of length 2. It then removes sequences that don't occur frequently and repeats the process by taking all of the remaining sequences and adding either a prefix activity or suffix activity to consider all sequences of length 3 and then removing those sequences that don't occur frequently. The process repeats either a fixed number of interactions or until no new sequences are found. To address interleaving as well as activity re-ordering, DVSM uses a Levenshtein (edit) distance to determine if two sequences are just variations on the same pattern. However, the authors fail to mention what they do if a newly discovered sequence is a variation of multiple previously discovered se-

quences. Frequency is based on the minimum description length principle [87] which argues that the best description of a data set is the one that maximally compresses it. Thus, the authors use the following inequality: $\frac{DL(D)}{DL(a)+DL(D|a)} > C$ where C is a minimum compression value, D is the dataset, a is a potential sequence and DL is the number of bytes to encode the argument. Once the sequences have been discovered, the authors use CLUSEQ [113] to categorize them.

Open Research Questions in Sequence Pattern Mining.

Gold [38] has shown that while it is possible to create a finite state automata from any fixed length sequence input, finding the minimum FSA is an NP-hard problem. Therefore, all gains in algorithms for discovering CFGs and regular grammars are incremental gains. Work continues in this area using different algorithms [76, 88, 25] and applying them in different domains [101, 4, 90]. Mooney [72] provides an overview of different approaches and algorithms.

One issue that remains is the inability to encode additional information in the transitions. For instance, if instead of `create`, `create(file1)` was desired, a transition could be created for it. However, if a `create(file2)` transition was also desired, there would need to be a second transition for it. While it is possible to encode `create` to reference an implicit file, e.g., `create(generic file name)`, there is no way to save that information for checking in future transitions, e.g., `create(file1) + access(file1)` and `create(file2) + access(file2)`. Walkinshawall, et al. [106] describe an extended finite state machine (EFSM) where transitions include data variables. While this does address the issue of memory, it does not appear to avoid creating (at least implicitly) transitions for each different data state.

Table 1. Acquisition Tools.

Tool	Artifact
Forensic Toolkit (FTK)	Hard Drive Images
Encase	Hard Drive Images
RamCapturer	Volatile Memory Images
WinDD	Volatile Memory Images
TheSleuthKit	File Times and Keywords
WevtUtil	Windows Event Logs
RegRipper	Windows Registry

2.2 Digital Forensics

Digital forensics is defined as “the discipline that combines elements of law and computer science to collect and analyze data from computer systems, networks, wireless communications, and storage devices in a way that is admissible as evidence in a court of law.” [67]. Over the past forty years [34], digital forensics has been adopted more and more by law enforcement agencies. In 2010, Gogolin stated that 50% or more of law enforcement agencies have added a digital components [37] and that number has undoubtedly increased in the past six years. As digital devices and the Internet continue to penetrate more areas of business and personal use, there will continue to be more of a need for digital forensic laboratories (DFL). Unfortunately, these DFLs are being overwhelmed with cases. James, et al. [52] observed in 2014 that there were reports of backlogs in digital forensic laboratories from 12 to 18 months and in some instances “approaching or exceeding 2 years.” Overill [78] and Shaw and Browne [94] suggest the problem is just getting worse as globally forensic labs are failing to keep up with demand. The problem is exacerbated by the volume of data increasing faster than processor and forensic tools can handle it [89]. Quick and Choo [82] summarize many experts who agree that demand exceeds supply and the gap is getting bigger.

Acquisition.

Digital Forensics includes acquisition, analysis and reporting[21]. As Carrier [17] observes, there are multiple layers of digital forensic acquisition. Table 1 shows some common tools used for acquiring different digital forensic artifacts. At the lowest level, there is simply the acquisition of the file system and volatile memory. Acquiring the data off of a physical hard drive is often as simple as mounting it on a different computer and then copying the data on it byte by byte. However, in cases with encrypted hard drives, simply acquiring the drive is of limited utility. In those cases, it is necessary to also acquire the encryption key, either from the hard drive user or from volatile memory while the hard drive is unlocked [42] or even shortly after power has been removed [41].

Acquiring volatile memory is more difficult than storage data. To acquire volatile memory, the user must interact with the memory without turning off the machine. Unless there is already hardware running that allows copying memory without interacting with the central processing unit (e.g., Firewire direct memory access tools [12, 62]), it's necessary to either attach a physical device or start a software application. Unfortunately, doing either modifies volatile memory. Since starting a software application is often the only choice, there are several software applications available [65, 99].

At the next level of acquisition, extraction tools look at individual file system and memory artifacts. The core functionality of The Sleuth Kit (TSK) [16] allows an analyst to analyze volume and file system data. TSK is often used to either look at unallocated/allocated space and search for relevant keywords or to view the meta information associated with the file system and the individual files [48, 32, 15]. There have been several tools that provide a wrapper around TSK to provide additional functionality [33, 31, 112]. Additional plugins allow analysis of file contents. Log2Timeline [40] incorporates TSK and extracts temporal data using plugins to

facilitate extracting information from many different types of files. While not an extension of TSK, Olsson and Boldt's Cyber Forensic Time Lab [77] also extracts time data from the file systems and several other file types, including the Windows registry.

Artifacts.

There are several file types that contain useful information for forensic examiners. First are log files from web browsers, e.g., Google Chrome, Microsoft Internet Explorer, and Mozilla Firefox. The Internet is used by most people to gather information and connect with the wider community and a web browser is a common tool for accessing it [74]. Analyzing where an individual went on the Internet and when can be very helpful in answering forensic questions. A second forensically interesting type of file is associated with office automation applications including Microsoft Office applications such as Microsoft Word, Excel, and PowerPoint [44]. If an individual is recording personal information and he is not doing it via a web browser, he is most likely using some office automation tool. The third file type of interest to digital forensic examiners includes image and video files. These are the most common types of files to either create, modify or view content downloaded from the Internet.

The last file type of interest are the files the operating system uses to perform its functions. Configuration files of interest include password files, user and group files, scheduled jobs, event logs, and applications [67]. Two types of Microsoft Windows operating system files with relevant temporal information include event logs and the Windows registry. The Windows event logs [68] capture application, security and set up events. These events vary from informational to warnings to errors to audits. The tool used to parse these logs is wevtutil [69] and is provided with the system install in the `textttc:/windows/system32` folder.

Table 2. RegRipper Plugins relevant to Temporal Artifacts.

Plugin	Registry Key	Artifact
Shellbags.pl	SOFTWARE/ MICROSOFT/ WINDOWS/ SHELL/ BAGMRU	Folder views (e.g., changing how files are displayed in Explorer)
ComDlg32.Pl	CURRENTVERSION/ EXPLORER/ COMDLG	Most recently opened files by extension
UserAssist.pl	SOFTWARE/ MICROSOFT/ WINDOWS/ CURRENTVERSION/ EXPLORER/ USERASSIST	Last launch time of GUI-based programs
RecentDocs.pl	SOFTWARE/ MICROSOFT/ WINDOWS/ CURRENTVERSION/ EXPLORER/ RECENTDOCS	Recently accessed documents in Windows Explorer by file type
WordWheelQuery.pl	SOFTWARE/ MICROSOFT/ WINDOWS/ CURRENTVERSION/ EXPLORER/ WORDWHEELQUERY	Recently used search words in Windows Explorer
NetworkList.pl	SOFTWARE/ MICROSOFT/ WINDOWS NT/ CURRENTVERSION/ NETWORKLIST/ PROFILES	When networks were connected to the computer
USBDevices.pl	SYSTEM/ {Current Control Set}/ ENUM/ USB	When USB devices were last connected to the computer
RemovDev.pl	SOFTWARE/ MICROSOFT/ WINDOWS PORTABLE DEVICES/ DEVICES	When USB devices were last connected to the computer
MountDev2.pl	SYSTEM/ MOUNTED DEVICES	When devices were connected to the computer
MountPoints2.pl	SOFTWARE/ MICROSOFT/ WINDOWS/ CURRENTVERSION/ EXPLORER/ MOUNTPOINTS2	When devices were connected to the computer

The Microsoft Windows Registry is a collection of files assembled to look like a file system [18]. This registry is used to store configuration information for the Windows Operating System as well as for individual applications. There has never been a systematic public effort to identify all of the information residing within the registry. In part, this is because much of the information is application (and O/S version) specific and, thus, such an exhaustive effort is unreasonable. There have been efforts to identify some of the information most useful to digital forensic examiners [19]. Harlan Carvey has developed a tool called Regripper [20] to extract information from the registry. Over time, others have added custom plug-ins to retrieve forensic items from the registry. Those plugins that provide relevant temporal information are shown in Table 2.

In addition, to the information stored on the hard drive, there is significant information available in volatile memory. Compiled Memory Analysis Tool (CMAT) [75], Volatility [108] and Rekal [85] are tools that provide an analyst with information extracted from volatile memory. Since everything that executes or is parsed resides in volatile memory at some point, it is possible to find a wealth of information including malicious code and disk decryption passwords. Other information available in volatile memory includes network connections to other devices, the current contents of the

clipboard, the current values in the Registry (which may differ from what's stored on the hard drive) and the active processes. Information for the active processes includes the current registry keys and files they are using, the user running the processes and the current contents of any files being modified. While accessing the contents of the particular applications requires an in-depth knowledge of the specific application running, the other forensic artifacts are easily retrieved. If the only thing digital forensic examiners are interested in are the contents of one or more specific files, this level of abstraction is sufficient. However, if they are interested in looking for patterns across multiple files, the information extracted needs to be abstracted further.

Timeline Reconstruction.

BackTracker [55] is one of the earliest tools to use processes, files, and filenames as items and process management and file system related calls as activities to reconstruct a sequence of activities related to an attack. Similar, Forensix [36] proactively intercepts Linux kernel network, process management, and file system related calls and then uses them to perform digital forensic analysis. More recently, Liao and Langweng [61] have expanded on previous work to include memory management, security and drivers. Khan [54] performs similar research but using data more like that proposed for the current research, including log files, registry entries, and file systems. He collects data in one second intervals from a computer and then feeds it a neural network and Bayes classifier to classify file system activities, e.g., Internet Explorer starting. The features he extracts are the five largest file sizes created, the five most frequent file accesses and the five most frequent registry key accesses during the 1 second window. Although the results of the above research are encouraging, they require sensors to be installed on the machine in advance of the event. To date, there has not been any research on performing the same level of abstraction based on data

collected after the incident occurs. Kalber [53] uses Levenshtein edit distances [60] to cluster similar paths and file names and then show the clusters visually based on timestamp proximity.

While there hasn't been much application of finite state automata to digital forensics, there has been some. James, et al. [51] and Gladyshev and Patel [35] use FSAs to encode evidence and witness statements and then reason about the case. The only examples provided in the papers are informal ones, e.g., trying to reconstruct who is using a shared printer based on witness statements and data collected from the print queue. Unfortunately, to date, there has not been any work done on automating the creation of finite state automata from existing sequences.

While this research addresses the overwhelming amount of digital artifacts by abstracting them, there are alternatives. One noteworthy option is to develop visualization tools that allow humans to parse the data more easily. Buchholz and Falk [13] developed Zeitline in 2005 as a Graphical User Interface (GUI) based tool that imported file system data from TSK and allowed searching and filtering of activities. Zeitline also allows users to combine low level activities into more complex activities. More recently, Hargreaves and Patterson developed Python Digital Forensic Timeline (PyDFT) [43] to perform low level event extraction and high-level event reconstruction. It begins by extracting times from the file system as well as inside of certain other file types including the Windows Registry and several well known web browsers. The data is stored in an SQLite database composed of three tables, info, events and keydata where the majority of the data is stored in the events table. They then use custom designed test events to summarize the low level events into complex events.

Finally, in addition to reconstructing what has occurred, it is important to store the digital artifacts extracted from multiple sources in a standard way. Carrier [17] describes a general framework for storing digital artifacts but does not provide details.

Chabot, et al. [22] extends this by providing a specific model titled the Semantic Analysis of Digital Forensic Cases (SADFC). It begins with a collection of events, $E = \{e_1, e_2, \dots, e_i\}$. Each event includes subjects which include human actors and processes and objects which includes, but are not limited to, webpages, files, and registry keys. Events take place in a time interval defined by a start time and end time. Chabot then goes on to describe relations between subjects and objects (e.g., participation, repercussion, creation, suppression) as well as event relations (e.g., causality and composition). Chabot, et al. [23], apply the SADFC and develop a system that begins with an extracting layer which acquires the data from seized images. The system then proceeds to a knowledge layer where the data extracted is placed into an ontology and organized to enable filtering and traceability. Third is the reasoning layer that analyzes and interprets the knowledge and finally there is an interface layer that provides visualization and querying tools. Turnbull and Randhawa [103] provide a comprehensive overview of previous work on ontologies and create a new comprehensive system, ParFor (Parallax Forensics) which includes extraction and uses forward chaining inference rules to develop abstractions.

2.3 Summary

As digital device become more integrated into our lives, more of our everyday activities are integrated into them. Fewer people are writing letters or developing photos. Instead, they use their digital devices for these activities. When something bad happens and digital forensic examiners need to reconstruct the activities leading up to it, gathering information from digital devices is becoming a requirement. Digital forensic examiners care about a quick reconstruction of a narrative of user activity related to the digital artifacts available on the device.

To date, most of the research in digital forensics has focused on acquiring data and

extracting the digital forensic artifacts of interest. Where higher level abstraction has occurred, the focus has been specifically on digital devices involved in cyber attacks. The general question of reconstructing a high level narrative of activities for non-cyber attack incidents has not been studied.

Abstracting temporal activities requires that the sequence of activities be maintained. When general pattern recognition techniques are applied to classifying sequences, they perform less well. While it is certainly possible to adopt them to mining sequential patterns, using techniques specifically designed for sequences might make more sense. Regular grammars are designed to describe sequential activities. While there has been much work in regular grammars, research in applying them to digital forensics has been limited. One challenge that regular grammars face is interleaving. Previous work with regular grammars has addressed noise but not specifically sequences occurring simultaneously.

Clustering is a proven technique for classifying items where there are no training labels [111]. It has previously been applied to categorizing sequences where the sequences themselves have already been identified. To date, no research begins by clustering temporally “close” items and then develops sequences from those clusters. Overall, this research looks to combine sequence pattern mining and regular grammars to construct human understandable activities from digital artifacts on a digital device.

III. Methodology

Digital forensic laboratories are overwhelmed by the number of cases they receive. While automation has helped reduce the time it takes to process the cases, most of the automation is limited to reducing the time to extract digital artifacts from digital media. Unfortunately, while data extraction is useful for finding individual pieces of information, it fails to automate the synthesis of digital artifacts into human understandable activities. If digital forensic examiners want to create a timeline of what has occurred on a computer and present it at a level understandable to lay people, there is no automated way to do it. This research utilizes sequence pattern mining to reconstruct and abstract temporal digital artifacts into human understandable activities. It creates sequences of digital artifacts using a factual baseline in a controlled environment and then applies them to a constructed dataset.

The remainder of this section begins by defining the research question and evaluation metrics and providing an overview of the experimental methodology. Next, a formal description of the domain is presented. Finally, the section concludes with high level descriptions of the algorithms.

3.1 Research Hypothesis and Evaluation Metrics

The research hypothesis is that sequence pattern mining can be used to automate abstracting digital artifacts into human activities. The research is evaluated by two metrics:

- Does the research provide an automated method for generating a set of sequences and production rules that enables describing with a single rule
 - starting and using Microsoft Office applications including Word, PowerPoint, and Excel

- starting and using web browsing applications including Chrome and Firefox
- How successful are the sequences and production rules at identifying occurrences of the above applications. Given a similarity threshold set such that all true positives are identified, is the false positive rate sufficiently low to allow practical use?

3.2 Method

The experiment is performed in five steps. First, a dataset is constructed. This crafted dataset is composed of five use cases with between six and eight scenarios for each use case. By manually constructing the dataset, the exact times that the human activities start and stop are known and used to construct the terms, sequences, and production rules. Once the dataset is constructed, the digital artifacts are extracted using a custom script (Appendix E). Third, the digital artifacts are examined to determine what the constraints on names to differentiate sequences should be. Fourth, the digital artifacts are used to create terms and sequences. Fifth, the terms and sequences that occur between the start and stop times of a user activity are consolidated into a set of production rules that describes that user activity. Finally, these terms, sequences and production rules are tested using a “leave one out” methodology to quantify how well they generalize.

3.3 Formal Description

Consider a collection of temporally ordered events that occur on a computer. Each event, e , is a set of characteristics, $\{\tau_s, \tau_e, v\}$ where τ_s and τ_e are the start and end times of the event and v is a digital artifact resulting from the event. The digital artifact, $v \in V$, consists of two parts ($v = (t, i)$), the operator, t , that performs the operation and the digital object(s), i , on which the operation is per-

formed. The operator, t , is a pair (s, r) composed of s , the actor (either the user application or the system if no user was involved) that performs the action, and r , the action performed on the set of digital objects, i , where each digital object is an organized collection of bits that provides content. Similarly, a digital object, i , is a set of zero or more objects on which the actor acts. Notationally, $e = \{\tau_s, \tau_e, V\} = \{\tau_s, \tau_e, (t, i)\} = \{\tau_s, \tau_e, ((s, r), \{i_1 \dots i_n\})\}$. For example, the set $\{10:00:00, 10:00:01, (\text{Windows Explorer}, \text{Create}), \{\text{"c:\temp\test.txt"}\}\}$ represents the event of Windows Explorer being used to create the file `test.txt` at 10:00 am. Observe that this representation is flexible enough to describe events with a single digital artifact (e.g., creating a single file) and events where one actor acts on multiple objects (e.g., backing up multiple files in a directory).

When a new file is created, several digital artifacts are created. First the file container is created (**Create**) and then immediately modified (**Modify**) with the new file contents (e.g., from a copy). Next, the file is accessed (**Access**) and the meta data in the master file table is changed (**cHange**). Using the above notation, these digital artifacts are represented as $((\text{Windows Explorer}, \text{Create}), \{\text{c:/temp/test.txt}\})$, $((\text{Windows Explorer}, \text{Modify}), \{\text{c:/temp/test.txt}\})$, $((\text{Windows Explorer}, \text{Access}), \{\text{c:/temp/test.txt}\})$, and $((\text{Windows Explorer}, \text{cHange}), \{\text{c:/temp/test.txt}\})$. If we represent the artifacts with the letters C (**Create**), M (**Modify**), A (**Access**), and H (**cHange meta**) respectively and use $a_1 + a_2$ to represent that digital artifact a_2 follows digital artifact a_1 , then we can represent this as $\text{CreateFile}(i_1) = C(i_1) + M(i_1) + A(i_1) + H(i_1)$ where $i_1 = \text{c:/temp/test.txt}$. For the purposes of this research, this is called a *sequence*. Since there is only one digital object, the sequence is called a *single-object sequence*. Similarly, a sequence that contains two or more objects, e.g., $C(i_1) + C(i_2) + M(i_1) + A(i_1) + H(i_1)$, is called a *multi-object sequence*.

The formal representation is as follows:

- S = the set of all actors, $s \in S$
- R = the set of all actions, $r \in R$
- I = the power set of all digital objects, $i \in I$
- T = the set of all operators, $t \in T$, where an operator is a
actor, action pair(s, r) s.t. $s \in S, r \in R$
- V = the set of all digital artifacts, $v = (t, i) \in V, t \in T, i \in I$
- $+$ = a binary non-commutative operator on V where
 $v_a + v_b$ is read as “ v_a is followed by v_b ”
and $v_a, v_b \in V$
- $\cdot$ = a non-commutative binary operator where
 $v \Leftarrow t \cdot r$ is read as “digital artifact v
results when actor t acts on digital object i ”
s.t. $v \in V, t \in T, i \in I$
- F = the set of functions that map a set of object
sets to a digital artifact, e.g., $f(\{i_1\}, \{i_2\}, \dots \{i_n\})$
 $= v_1(t_1, I_1) + v_2(t_2, I_2) + \dots + v_n(t_n, I_n)$
s.t. $t_1, t_2, \dots, t_n \in T, i_1, i_2, \dots, i_n \in I$,
 $I_1, I_2, \dots, I_n \subseteq \{i_1, i_2, \dots, i_n\}$
 $v_1, v_2, \dots, v_n \in V$
- ϵ = the number of events
- τ_s = the start time of an event
- τ_e = the end time of an event

E = the set of events, where an event is a start time,
 end time, digital artifact triplet (τ_s, τ_e, v)
 s.t. $\tau_s, \tau_e \in \mathbb{N}, v \in V$
 $\leq$ = a binary relational operator indicating temporal
 precedence written $e_i \leq e_j$
 which is true if $\tau_s^i < \tau_s^j$ or if $\tau_s^i = \tau_s^j$ and
 $\tau_e^i \leq \tau_e^j$
 e_{01} = the event where $\forall_{e \neq e_{01} \in E} e_{01} \leq e$
 e_{02} = by similar logic $\forall_{e \neq e_{01}, e_{02} \in E} e_{02} \leq e$
 e_{0k} = $\forall_{e \neq e_{01}, e_{02}, \dots, e_{0k} \in E} e_{0k} \leq e$
 E_0 = the set of events that occurred on the digital device
 and are represented as $\{e_{01}, e_{02}, \dots, e_{0\epsilon}\}$

Attaching Variables to SPM algorithms.

Variables provide receptacles for attributes of mined sequences without the need to change the SPM algorithm itself. For instance consider a sequence of computer component purchases where if a customer purchases a memory card and storage, about 15% of the time he later purchases a video card (e.g., $S2 = \text{memory} + \text{storage} + \text{video card}$). That may be statistically significant, but does it merit targeted advertising? If a **price** attribute is added to each item, then analysis of the mined sequences may show that when the price of the memory card and storage are both above \$100, the purchase of the video card occurs 80% of the time. Using variables as a receptacle to store attribute values allows better analysis of the mined sequences.

Variables also enable sequences based on the values of the attributes. In the previous example, variables enable sequences like $\text{memory}(\text{price} > \$100) + \text{storage}(\text{price} > \$100) + \text{video card}$. In this case, when the SPM mines a sequence, it passes it off to a *variable attachment* algorithm for further processing. The result is that sequences that

have the same items (e.g., $S1 = \text{memory} + \text{storage} + \text{video card}$) may become sequences with different terms after variable attachment (e.g., $S2(p, q | p, q > 100) = \text{memory}(p) + \text{storage}(q) + \text{video card}$ and $S2(c, d | c < 50, d < 80) = \text{memory}(c) + \text{storage}(d) + \text{video card}$). For consistency sake, the variable attachment algorithm pre-defines variables with different constraints (e.g., p and q are constrained to always be greater than 100, while c is always less than 50 and d is always less than 80). Thus, whenever these variables are assigned to attributes, it is clear what the constraints for those attributes are.

While the previous examples have been restricted to a single attribute, attaching variables to SPM generalizes to multiple attributes. For instance, rather than limiting the attributes of the **memory**, **storage** and **video card** items to price, an additional attribute of size can be added. Perhaps it is not the price of memory and storage that is influencing whether video cards are part of the sequence but rather the size of memory and storage. Or, perhaps by adding in a variable with the size attribute to the video card, we can find an additional relationship between the price of memory and storage and the size of the video card. For simplicity, the remainder of this research only considers items with a single attribute that can have a variable number of values; however, everything discussed generalizes to multiple attributes without loss of generality.

As shown in Figure 3, sequence mining of terms takes several steps. First the sequence of terms is identified. Then the number of variables needed is determined by considering the different attribute values in the different terms. Next, variables are assigned to the different values based on the categories the values fall into. Finally, the sequence with the variables assigned is fed back to the SPM for further processing. Observe that in Figure 3 that sequence S_1 is transformed into sequence $S_{1(a,f)}$. In other words, different instances of the same sequence of terms may transform into

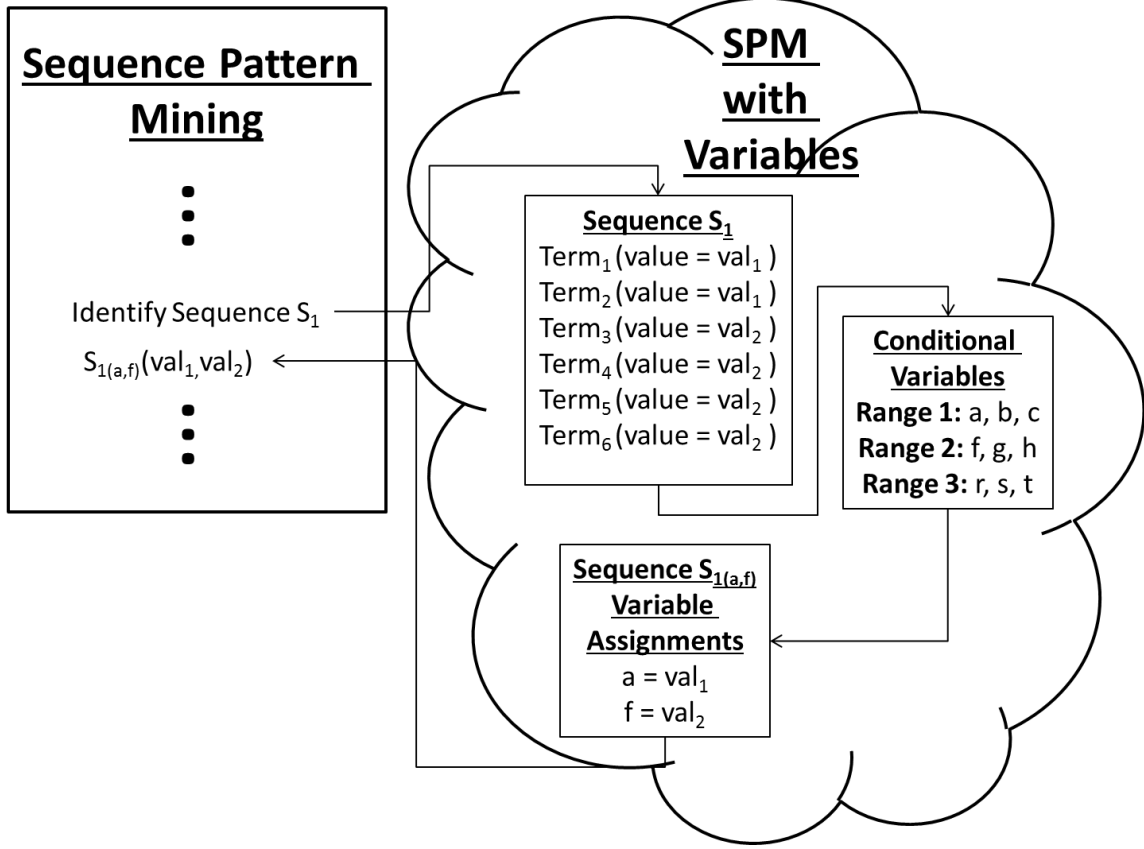


Figure 3. Attaching Variables to SPM.

different sequences (e.g., $S_{1(a,f)}$, $S_{1(a,b)}$, or $S_{1(f)}$ based on the number and types of different values the attributes of those terms have.

Sequence Pattern Mining with Variables can handle mining sequences of terms where the terms are abstractions. Consider a sequence A, B, C . A might be `memory1 ($120)`, B might be `memory2 ($123) + memory3($130) + memory4($127)`, and C might be `memory1($120) + storage1($150) + video2($124)`. A is a simple term. B is a loop and C is a sequence of terms, each with one or more variables. The process described in Figure 3 becomes more involved. As shown in Algorithm 1, there are now three possibilities when considering a term (i.e., simple terms, loops, and sequences). Simple terms are handled as above. Loops are handled like simple terms except that there is no attempt to find other terms in the sequence with the same list of values.

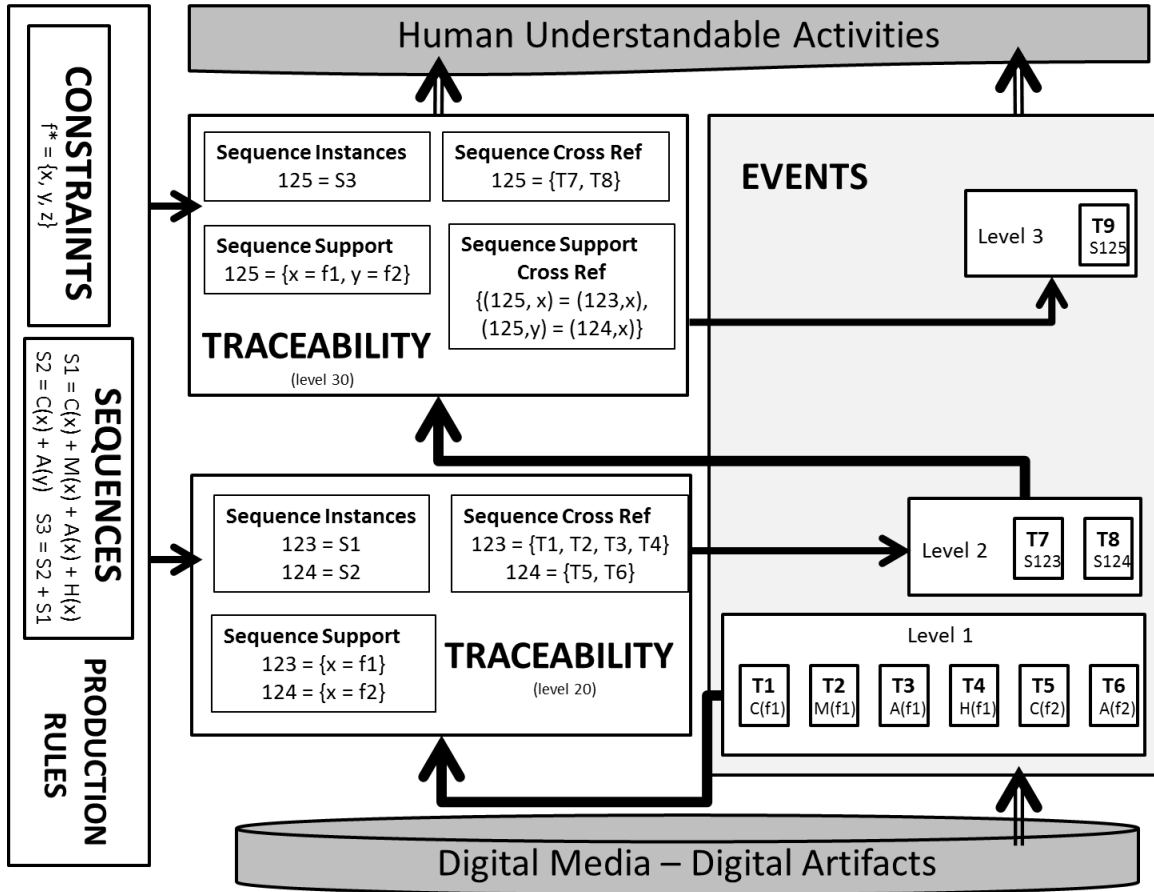


Figure 4. Traceability.

However, sequences, which already have a list of variable assignments, need additional consideration. As shown in Figure 4, these variable assignments need to be combined with the variable assignments for the other items in the new sequence. When the same value has already been assigned to a different variable, a cross reference between the new variable and the old variable needs to be made for traceability. When the variable assignments for a sequence is completed, it has the same structure whether it contains sequences or not. The only difference is the cross reference used for traceability.

Algorithm 2: SPM - Variable Attachment (part 1).

Input: S_i - a sequence of terms $ConditionalVariables$ - list of conditions and a list of variables for each condition**Output:** S_o - a sequence of terms X_v - cross reference of variables $Used_v$ - list of used variables**Variables:** t_{new} - a copy of S_i 's current term v - variable being assigned to current term $attList$ - list of attribute values associated with term $wattribute$ - attribute value being examined for term $Xref$ - if term is a sequence, cross reference τ_1 - the end time for the first item in the workspace for the instance of that sequence in the dataset v_{old} - variable/value for an attribute in $Xref_i$ v_{new} - variable/value for an attribute for S_o

3.4 Constraint Generation

For this research, the constraints are based on the names of the digital objects. Digital object names are a collection of words separated by slashes where the last word is the digital object name and the preceding words are the folders and sub-folders it is stored in (e.g., (windows, system32, config, software) and software, microsoft, office, 14.0, word, file mru). A pattern is an ordered sequence of words with zero or more of the words replaced with a wildcard (e.g., (windows, system32, config, *), (windows, *, *, *), (*, *, *, *), (windows, *, config, software)). For simplicity and without loss of generality, adjacent wildcards are reduced to a single wildcard (e.g., (windows, system32, config, *), (windows, *), (*), (windows, *, config, software)). Constraining digital objects names in this way enables greater granularity in terms and sequences such that a review of the terms in a sequence is understandable to a subject matter expert.

Algorithm 3: SPM - Variable Attachment (part 2).

Algorithm:

$Used_v, X_v \leftarrow \emptyset;$

for $i \leftarrow 1, S_i.getLength()$ **do**

$t_{new} \leftarrow S_i.getTerm(i);$

$t_{new}.xref, t_{new}.variable \leftarrow NULL;$

if $t_{new}.isLoop()$ **then**

$v = ConditionalVariables.getVariable(NULL);$

$t_{new}.variable \leftarrow v;$

if $t_{new}.isSingleValue()$ **then**

$v = Used_v.getVariable(t.getAttributes());$

if $v == NULL$ **then**

$v = ConditionalVariables.getVariable(t.getAttributes());$

$Used_v \leftarrow Used_v \cup \{v, t.getAttributes()\};$

$t_{new}.variable \leftarrow v;$

else

$attList \leftarrow t.getAttributes();$

while $wattribute = attList.pop()$ **do**

$v = Used_v.getVariable(t.getAttributes());$

if $v == NULL$ **then**

$v = ConditionalVariables.;$

$getVariable(t.getAttributes());$

$Used_v \leftarrow Used_v \cup \{v, t.getAttributes()\};$

$Xref \leftarrow S_i.getTerm(i).getXref();$

for $j \leftarrow 1, Xref.getLength()$ **do**

$witem \leftarrow Xref.getAttributes(j);$

$v_{old} \leftarrow Xref.getVariable(wattribute);$

$v_{new} \leftarrow Used_v.getVariable(wattribute);$

if $v_{new} == NULL$ **then**

$v_{new} = ConditionalVariables.;$

$getVariable(wattribute);$

$Used_v \leftarrow Used_v \cup \{v_{new}, wattribute\};$

$X_v \leftarrow X_v \cup \{v_{old}, v_{new}\};$

$t_{new}.xref \leftarrow X_v;$

$S_o \leftarrow S_o \cup t_{new};$

There are several possible methods for determining the constraints for digital object names. The first is to have a subject matter expert (SME) exhaustively define every allowable pattern. This creates a significant demand on a SME and requires their presence each time a new operating system is encountered. The second is to have a subject matter expert create rules for allowable patterns (e.g., windows is acceptable in a pattern but a specific user name is not) and then create patterns on the fly. There are two issues with this. The first is that operating systems need to be split into classes based on similar file system structures. This cuts down SME involvement but still requires it sporadically. The second issue is that there is no way to know how many patterns will be needed until the entire dataset is encountered. If too many patterns are generated, the production rule generation and abstraction algorithms will take much longer to execute.

For this research, the constraints are created after the digital artifacts are loaded but prior to executing the sequence mining, production rule generation and abstraction algorithms. Due to limitations on the research machine, a threshold is set to restrict the number of patterns to around 400. Each digital object is reviewed and every possible pattern is created (e.g., (windows, system32, config, software) results in { (windows, system32, config, software), (windows, system32, config, *), (windows, system32, *, software), (windows, *, config, software), (*, system32, config, software), (windows, *, config, *), (*, system32, config, *), (windows, system32, *), (*, system32, *, software), (windows, *, software), (*, config, software), (*, config, *), (*, system32, *), (windows, *), (*, software), (*) }). Then the number of digital artifacts that match a pattern are counted. A pattern's score is the number of times it occurs multiplied by its character length. Patterns are kept if their score is above a threshold. While, this method would work best if all patterns were evaluated

against each other. However, due to the memory and processing power limitations, the set of digital artifacts were broken into approximately 60 subsets. The threshold for each subset is a percentage of the highest score for that subset. Thresholds vary from 0.05% to 7.00% and are manually adjusted to keep the total number of patterns to around 400. For each pattern that is kept, a variable is assigned to the pattern for later use in creating terms.

3.5 Term, Sequence and Production Rule Generation

Once the constraints have been defined, the terms and sequences are created from the digital artifact transactional data using the SPM algorithms. The first algorithm, Single Object Sequence and Loop Abstraction (SOSLA) is a greedy algorithm for mining sequences. The second, Discontinuous Varied Order Sequence Mining with Variables (DVSM-V), takes the DVSM algorithm [84] and extends it to the digital forensics domain. Finally, the third algorithm, Sequential Mining of Temporal Clusters (SMTC), generates clusters of temporally “close” items and then creates sequences from those clusters.

The process of creating terms, sequences, and production rules is shown in Figure 5. Running SOSLA, DVSM-V, and SMTC on the digital artifacts shows the sequences generated by DVSM-V and SMTC are more difficult than SOSLA-generated sequences to describe in human understandable terms. Since both DVSM-V and SMTC algorithms allow for multi-object sequences, the sequences often have two parts of one object and two parts of a second object. For instance, given the items $C(f1) + M(f1) + A(f1) + H(f1) + C(f2) + A(f2)$, SOSLA creates two sequences: $CMAH(f1)$ and $CA(f2)$; however, DVSM-V and SMTC are equally likely to produce $CM(F1)$ and $AH(f1) + CA(f2)$. Both sets of sequences are correct, but SOSLA’s is more understandable. As a result, SOSLA is used as a pre-processing step on the im-

ported digital artifacts and SMTC and DVSM-V are used on the SOSLA abstracted data. Appendix C details all of the parameters for the process. Two sets of terms, sequences, and production rules are created, one using DVSM-V and one using SMTC. As a result, the digital artifacts in E_0 are abstracted several times. At each stage, new sequences are created and then applied to create a more abstracted dataset.

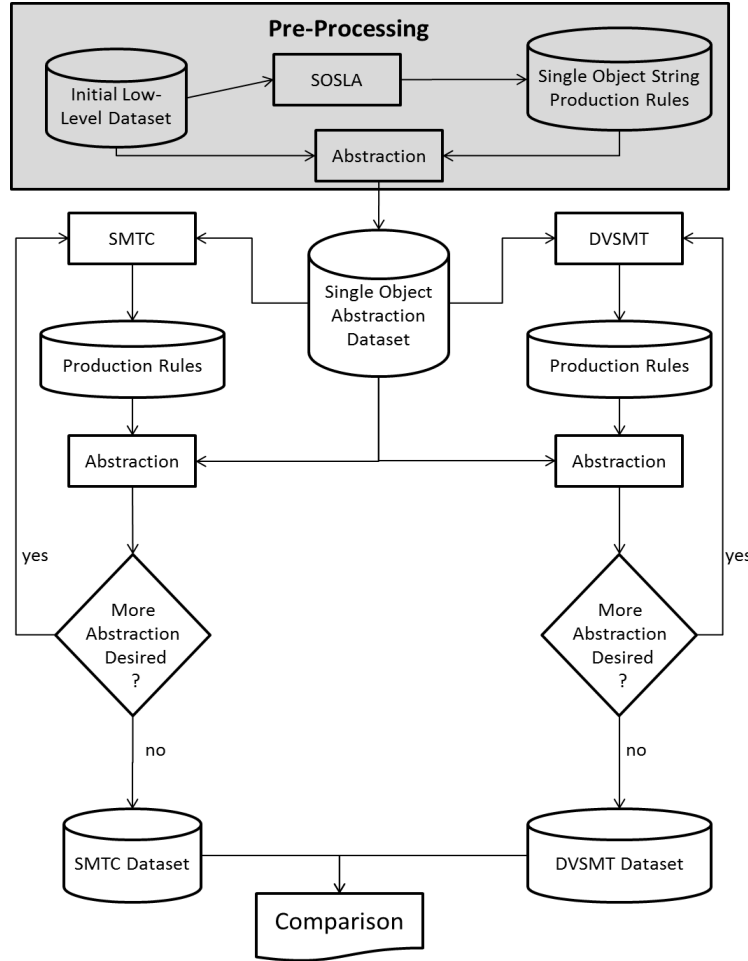


Figure 5. Abstraction Methodology.

Single Object Sequence and Loop Abstraction (SOSLA).

SOSLA is a greedy algorithm that makes a single pass through the data creating sequences as it goes. In its simplest form, SOSLA only creates sequences for single

digital object sequences. As shown in Algorithm 4, SOSLA examines each item sequentially. For each item, it checks which constraint its digital object matches. It then creates a term that replaces the item’s digital object with the variable assigned to the constraint’s pattern (in the case where the item is a sequence or loop, it uses the logic described in Algorithms 2 and 3). As soon as it comes across an item where the digital object is different than the previous item, it examines its current sequence of items. If this sequence contains at least two items and occurs within a user specified period of time, it checks whether a sequence for these items with the variable exists. If such a sequence does not exist, it creates it. It then removes the previous items, adds the new one and continues through the dataset until it reaches the end.

By removing the single term restriction, SOSLA becomes SOSLA-Multi Object sequence (SOSLA-M). In theory, if all the items in the dataset occur close enough temporally, SOSLA-M produces a single rule. If a maximum number of terms is set, SOSLA-M creates all sequences under that maximum length. If the time frame is restricted, but there is no maximum number of terms for a single rule, the result is a single sequence that completely describes all items that occurred during the time frame. In a controlled environment, this would result is a single sequence that completely describes what occurred during that time frame.

Discontinuous Varied Order Sequence Mining with Variables (DVSM-V).

DVSM-V is DVSM applied to the digital forensics domain through the addition of variables. Although DVSM has been described earlier, it is described here for ease of reference. As shown in Algorithm 5, DVSM proceeds sequentially through the dataset creating all possible sequences of size 1. It then saves those sequences it considers interesting, i.e.,

Algorithm 4: Single Object Sequence and Loop Abstraction (SOSLA).

Input:

D - the dataset - a sequence of items (e.g., E_0)

M_S - maximum number of terms in a rule

M_τ - maximum time lag between the first and last terms

Output:

S - the set of generated sequences

Variables:

W - the workspace - a subset of D currently being examined

e - the current item

w_i - the i^{th} entry in the workspace

i_1 - the digital object for the current item

i_o - the item from the previous item

τ_1 - the end time for the first item in the workspace

Algorithm:

$D.gotoBeginning();$

$S, W \leftarrow \emptyset;$

$\tau_1, i_o \leftarrow 0;$

while ($e \leftarrow D.nextItem()$) $\neq NULL$ **do**

$i_1 \leftarrow e.getObject();$

if ($distance(\tau_1, e.getTime()) \geq M_\tau$ **and** $\tau_1 \neq 0$) **or**
 (*only single object sequences* **and** $i_1 \neq i_o$ **and** $i_o \neq 0$) **or**
 (*multi object sequences* **and** $|W| \geq M_S$) **then**

$W' \leftarrow W;$

while $W' \neq \emptyset$ **do**

if $W' \notin S$ **then**

$S \leftarrow S \cup W;$

if *single object sequence* **then**

$W' \leftarrow \emptyset;$

else

$W' \leftarrow W' - w'_1;$

$W \leftarrow \emptyset;$

$\tau_1 \leftarrow 0;$

$W \leftarrow W \cup e;$

$i_o \leftarrow i_1;$

if $\tau_1 = 0$ **then**

$\tau_1 \leftarrow e.getTime();$

$$\frac{DL(D)}{DL(a) + DL(D|a)} > C \quad (1)$$

where C is a minimum compression value, D is the dataset, a is a potential sequence and DL is the number of bytes to encode the argument). It then looks at all of the interesting sequences and creates sequences of size 2 based on the items that occurred immediately preceding and following the size-1 sequences. Once it has created all of these size-2 sequences, it again removes all but those it considers interesting. This process repeats either until it does not consider any sequences of the new length interesting or until it reaches a previously defined maximum length. The algorithm concludes by producing sequences for all of the interesting sequences of size 2 or more. DVSM-V works identically to DVSM except that when it creates sequences, it uses the extensions described in Section 3.3.

Sequence Mining of Temporal Clusters (SMTc).

While there are several SPM algorithms that handle interleaving, most handle it as an afterthought. Interleaving occurs when two or more sequences of terms occur simultaneously. For instance, if sequence 1 is *ABBCD* and sequence two is *CBFFA* then *ACBBBCFFDA* is an example of those two sequences interleaved. If *ACBBBCFFDA* is the only example of sequences 1 and 2, then it's impossible to separate them. However, if they occur frequently elsewhere as individual sequences where there isn't interleaving, it becomes possible to find them in *ACBBBCFFDA*.

A common method for decoupling interleaved sequences is to compare the sequence being examined with previously extracted sequences. If the sequence in question is "close enough" to a previously extracted sequence, then it is considered another example of the sequence. There are several methods for determining "closeness". One measurement of distance that is often used is the Levenshtein edit distance [60]. This

Algorithm 5: Discontinuous Varied order Sequential Mining with Variables (DVSM-V).

Input:

D - the dataset - a sequence of items

M_S - maximum number of terms in a production rule

$threshold_M$ - the minimum score for a main sequence to be considered interesting

$threshold_V$ - the minimum score for a variant to be considered interesting

$threshold_S$ - the maximum Levenshtein edit distance between sequences for them to be considered variants

Output:

S_d - the set of generated sequences of length d (e.g., S_1 is all 1-sequences).

Variables:

e - the current item

s - a generated sequence and all the instantiations in D of it

s_1, s_2 - a new sequence to be generated

t - an instantiation of a sequence d - the current sequence length being examined

Algorithm:

$S_1 \leftarrow \emptyset$;

$D.gotoBeginning()$;

while ($e \leftarrow D.nextItem()$) $\neq NULL$ **do**

$s \leftarrow e.getTerm()$;
If $s \notin S_1$ **Then** $S_1 \leftarrow S_1 \cup s$;
 $s.instantiations.insert(e)$;

$S_1 \leftarrow resolveVariants(S_1, threshold_S)$;

$S_1 \leftarrow KeepInteresting(D, S_1, threshold_M, threshold_V)$;

for $d = 2 : M_S$ **do**

$S_d \leftarrow \emptyset$;
for $s \in S_{d-1}$ **do**
 for $t \in s.getInstantiations()$ **do**
 $s_1 = t.getPreviousItem().getTerm() + s$;
 $s_2 = s + t.getNextItem().getTerm()$;
 If $s_1 \notin S_d$ **Then** $S_d \leftarrow S_d \cup s_1$;
 $s_1.instantiations.insert(t.getPreviousItem() + t)$;
 If $s_2 \notin S_d$ **Then** $S_d \leftarrow S_d \cup s_2$;
 $s_2.instantiations.insert(t + t.getNextItem())$;

$S_d \leftarrow resolveVariants(S_d, threshold_S)$;

$S_d \leftarrow KeepInteresting(D, S_d, threshold_M, threshold_V)$;

Algorithm 6: DVSM-V - Resolve Variants.

Input:

S - the set of generated sequences.

$threshold_S$ - the maximum Levenshtein edit distance between sequences for them to be considered variants

Output: S - the set of interesting sequences.

Variables:

W - the workspace - a set of similar sequences

s, s' - a generated sequence and all the instantiations in D of it

s_m - the main sequence that a variant sequence is based on

Algorithm:

```
for  $s \in S$  do
     $s.isMain() \leftarrow \mathbf{FALSE}$  ;
     $s.setMainVariant(0)$ ;
for  $s \in S$  do
    if  $s.isMain() = \mathbf{FALSE}$  and  $s.getMainVariant() = 0$  then
         $W \leftarrow \emptyset$ ;
        for  $s' \in S$  do
            if  $s'.isMain() = \mathbf{FALSE}$  and  $s'.getMainVariant() = 0$  then
                if  $levenshteinDistance(s, s') < threshold_S$  and  $s \neq s'$  then
                     $W \leftarrow s'$ ;
        if  $W \neq \emptyset$  then
             $s_m = mostCenter(W)$ ;
             $s_m.isMain() = \mathbf{TRUE}$  ;
            for  $s \in W$  do
                if  $s \neq s_m$  then
                     $s.setMainVariant(s_m)$ ;
```

determines the minimum number of simple edit operations necessary to transform one sequence to another. In the above example, transforming *ACBBBCFFDA* into *ABBCD* or *CBFFA* would require five delete operations. Thus, the distance between the sequences is five. There are several issues with this. The first is determining what a reasonable threshold for closeness is. In the above example, five operations are required to transform *ACBBBCFFDA* into *ABBCD* and *CBFFA* (four if we can ignore the final or first characters respectively). Using that threshold would mean that sequence 1 and 2 are also the same sequence (since they can be transformed

Algorithm 7: DVSM-V - Keep Interesting.

Input: D - the dataset - a sequence of items S - the set of generated sequence. $threshold_M$ - the minimum score for a main sequence to be considered interesting $threshold_V$ - the minimum score for a variant to be considered interesting**Output:** S - the set of interesting sequences.**Variables:** s - a generated sequence and all the instantiations in D of it s_m - the main sequence that a variant sequence is based on $score$ - a score of how interesting a sequence is, i.e., how much information gain the sequence produces**Algorithm:****for** $s \in S$ **do**

```
    if  $s$  is a main sequence then
        score  $\leftarrow |D|/(|s| + (|D| - |s|) * s.getOccurrences());$ 
        if score  $< threshold_M$  then
             $S \leftarrow S - s;$ 
    else
        score  $\leftarrow (|D| - |s_m| * s_m.getOccurrences())/$ 
             $(|D| - |s| * s.getOccurrences());$ 
        if score  $< threshold_V$  then
             $S \leftarrow S - s;$ 
```

into each other with only 4 swap operations). One way around this issue is to put different costs on different types of operations (e.g., the delete operation costs 1 distance while the insert and swap operations cost 2). A second issue in separating interleaved sequences is that the same candidate sequence needs to be compared to every previously extracted sequence. For instance, it is not sufficient to simple transform $ACBBBCFFDA$ into $ABBCD$; it needs to be transformed into $ABBCD$ and $CBFFA$. Thus, once one sequence is extracted, the terms remaining in the candidate (i.e., the noise) needs to be examined again to see if there are more sequences hidden in it.

An alternative method for resolving interleaving is to focus first on what terms

occur together. By first clustering terms that occur close together, we avoid several complications. First, there is no need to worry about the order of terms varying (e.g., one time when making a sandwich, they get a knife out first and the next time, they get the bread out first). Terms are first considered without order, only as temporally close sets. Second, since temporally close terms are first placed in clusters, there is no concept of interleaving. Each time two or more temporally close terms occur, all possible subsets of those terms are placed in clusters. For instance, if v_1, v_2 and v_3 occur temporally close together, the subsets $\{v_1\}, \{v_2\}, \{v_3\}, \{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}$, and $\{v_1, v_2, v_3\}$ are all created. Then the existing set of clusters is checked to see if these subsets already exist. If they do, the new instance of the subset is added to the cluster. In this way, once the entire dataset has been examined, the resulting clusters are free from interleaving. Any cluster that exceeds a threshold (discussed below) is considered interesting. Only after the clusters have been created and the uninteresting ones removed is sequencing considered. Sequence Mining of Temporal Clusters (SMTC) determines if a cluster is interesting by using a score of

$$occurrences(C) * |C|^{variableCount(C)*\epsilon} \quad (2)$$

$Occurrences(C)$ is the number of times this cluster of terms occurs in the dataset. $VariableCount(C)$ is the number of distinct variables in C , and ϵ is a user specified weighting. Observe that this is a different formula than in DVSM-V, Equation 1. DVSM-V uses the formula used in DVSM, which does not account for variables. This new formula uses a weight to favor sequences with fewer variables. The second implementation decision is how to determine the optimal sequence for a given cluster. In this implementation, the sequence that occurred most frequently in the dataset (i.e., the mode) is considered the representative sequence for the cluster. However, other choices can be made (e.g, using the median sequence) without changing the SMTC

class of algorithms. Algorithm 8 shows the SMTC class of algorithms. Parameters associated with SMTC include the maximum number of terms in the workspace (i.e., how big of a power set can the machine handle), how much weight to give to fewer different variables in a sequence, the maximum time lag for terms to be considered temporally close, and the threshold score for SMTC to keep a cluster of terms.

3.6 Dataset Abstraction

Once the sequences have been created, they can then be applied to the dataset to create a smaller, more abstracted dataset with full traceability. Abstraction is done in two steps. First, the entire dataset is processed using the terms and sequences (sequence abstraction). Then, this interim dataset is processed a second time where the same term occurring two or more times in sequence is replaced with a loop (loop abstraction).

As shown in Algorithm 10, the sequence abstraction algorithm proceeds through the dataset sequentially adding items to the workspace until either the workspace is the size of the largest sequence or the next item is temporally too far away from the first item in the workspace. It then creates a term for each item in the workspace that replaces the item's digital object with the variable assigned to the constraint's pattern (in the case where the item is a sequence or loop, it uses the logic described in Algorithms 2 and 3). Next, it attempts to find a sequence for the workspace. It begins with all of the items in the workspace and if it fails to find a rule that matches, it removes the most recent item and repeats until it either finds a match or is left with only a single item. If a sequence is found, a new item is created in the interim dataset with the associated items replaced with a single sequence symbol with traceability to the input dataset. Otherwise, the single oldest item is copied to the new interim dataset. The moved item(s) are then removed from the workspace and

Algorithm 8: Sequence Mining of Temporal Clusters (SMTTC).

Input:

D - the dataset - a sequence of items

M_S - maximum number of items in a sequence

M_τ - maximum time lag between the first and last items

ϵ - the weight of having more items in a sequence

threshold - the minimum score necessary to create a sequence

Output:

s_i - a generated sequence and all the instantiations of it

S - the set of generated sequences, $\cup s_i$

Variables:

e - the current item

W - the workspace - a subset of D currently being examined

w_i - the i^{th} entry in the workspace

T - the power set generated by W where each $t \in T$ has all its instantiations attached

C - the set of clusters. For $c \in C$, c is an unordered set, but the instantiations attached to c are ordered as in D

Algorithm:

$S, W, C \leftarrow \emptyset$;

$D.gotoBeginning()$;

while ($e \leftarrow D.nextItem()$) $\neq NULL$ **do**

if ($|W| > M_S$ **or** $distance(w_1, e) > M_\tau$) **then**

$T \leftarrow powerset(W)$;

for $t \in T$ **do**

$c \leftarrow t$;

if $c \in C$ **then**

$c.occurrences++$;

$c.addInstantiations(t.getInstantiations())$;

else

$C = C \cup c$;

$c.occurrences \leftarrow 1$;

$c.addInstantiations(t.getInstantiations())$;

 Remove items from workspace till new item fits in workspace;

$S \leftarrow KeepInteresting(C, \epsilon, threshold)$;

Algorithm 9: SMTTC -Keep Interesting.

Input: C - the set of generated clusters ϵ - the weight of having more terms in a sequence $threshold$ - the minimum score necessary to create a sequence**Output:** s_i - a generated sequence and all the instantiations of it S - the set of generated sequences, $\cup s_i$ **Variables:** $score$ - a score of how interesting a sequence is, i.e., how much information gain the sequence produces**Algorithm:** $S \leftarrow \emptyset;$ **for** $c \in C$ **do** $score \leftarrow (c.getOccurrences()) * |c|^{c.getVariableCount()*\epsilon};$ **if** $score < threshold$ **then** $C = C - c;$ **else** $s \leftarrow getMostFrequentSequence(c.getInstantiations());$ $S = S \cup s;$

the process repeats until the entire dataset has been examined. Creating loops is also straightforward. As show in Algorithm 11, the dataset abstracted by the sequence abstraction algorithm is examined and when two or more adjacent items are found with the same operator, they are replaced with a loop.

3.7 Rule Consolidation

In most research, when multiple sequences that describe the same thing are clustered, the centroid of the cluster is used as the best descriptor. However, in this research, the uniqueness of each sequence needs to be preserved while consolidating similar sequences into a unified descriptor of the use case. For instance, a single sequence for starting Microsoft Word is needed that preserves the differences observed by starting it in different ways. A literature review provided no algorithms that accomplish this.

Algorithm 12 accomplishes this by treating each sequence as a collection of terms.

Algorithm 10: Sequence Abstraction.

Input:

D - the dataset - a sequence of items

M_S - maximum number of terms

M_τ - maximum time lag between a the first and last terms

Output: A - an abstracted dataset with traceability back to D

Variables:

e - the current item

W - the workspace - a subset of D currently being examined

T - a subset of the workspace being examined to see if there is a sequence for it

w_i - the i^{th} entry in the workspace

Algorithm:

$A, W \leftarrow \emptyset$;

$D.gotoBeginning()$;

while ($e \leftarrow D.nextItem()$) $\neq NULL$ **do**

if ($|W| > M_S$ or $distance(w_1, e) > M_\tau$) **then**

for $i = |W| : 1$ **do**

if $i = 1$ **then**

$A = A \cup w_1$;

 Create traceability;

$W = W - w_1$;

 Exit For Loop;

$T \leftarrow \{w_1, w_2, \dots, w_i\}$;

if $T \in P$ **then**

$A = A \cup P(T)$;

 Create traceability;

$W = W - T$;

 Exit For Loop;

$W = W \cup e$;

Algorithm 11: Loop Abstraction.

Input: A - an abstracted dataset M_τ - maximum time lag between the first and last terms**Output:** B - an abstracted dataset with traceability back to D **Variables:** e - the current item W - the workspace - a subset of D currently being examined w_i - the i^{th} entry in the workspace o_o - the previous operator e_o - the current item's operator**Algorithm:** $B, W \leftarrow \emptyset;$ $o_o = 0;$ $A.gotoBeginning();$ **while** ($e \leftarrow A.nextItem()$) $\neq NULL$ **do** $e_o \leftarrow e.getOperator();$ **if** $|W| > 0$ **then** **if** ($dist(w_1, e) > M_\tau$) **or** e is a loop **or** $e_o \neq o_o$ **then** **if** $|W| > 1$ **then** $B = B \cup CreateLoop(W);$ **else** $B = B \cup w_1 ;$ $W \leftarrow \emptyset;$ $W = W \cup e;$ $o_o \leftarrow e_o;$

Even if these terms are themselves sequences, they are treated as terms. The sequence for the first use case is transformed into a directed acyclic graph where each vertex is a term and each edge is a connection between adjacent terms. The sequence for each subsequent scenario is then folded into the graph as follows: the graph and sequence are simultaneously traversed to find the matching vertex and term that are closest in position to the front of their respective lists (e.g., vertex 3 represents term 123 and the third term in the sequence is term 123). The term is assigned to the vertex and the process is repeated on the smaller left and right lists. If no vertex and term are found that match, the remaining terms in the sequence are then added to the graph.

Algorithm 12: Sequence Consolidation.

Input:

T - an ordered list of terms where t_i represents one set of terms

S - A set of lists of terms, i.e., $S = \{t_1, t_2, \dots, t_n\}$

Output: P - the set of production rules needed to produce the inputted sets of terms

Variables:

t_0 - the next set of terms

$sVertex$ - a null Vertex that is the source of the graph

$eVertex$ - a null Vertex that is the sink of the graph

G - a graph that satisfies all of the inputted sequences of terms

Algorithm:

$G, P \leftarrow \emptyset; S.gotoBeginning();$

$t_0 \leftarrow S.nextSetofTerms();$

$AddVertices(G, t_0); createVertex(sVertex, NULL);$

$createVertex(eVertex, NULL); G \leftarrow sVertex \cup G \cup eVertex;$

while ($t_0 \leftarrow S.nextSetofTerms() \neq NULL$) **do**

$\lfloor buildGraph(G, t_0, sVertex, eVertex, t_0.getFirstTerm(), t_0.getLastTerm());$

$P \leftarrow G.generateRules();$

A natural way of adding the new terms is to create new vertices for these terms and then create a new directed path between the two endpoints. When attempting to implement this method, the complexity of the search for matching vertices and terms and the subsequent complexity of traversing the graph to determine if the graph matched the inputted sequences of terms increased exponentially. This method was

Algorithm 13: Build Graph.

Input:

G - a DAG where the vertices are terms and the edges are adjacency

T - an ordered list of terms where t_i represents one term

$ep1$ - the first vertex in the graph

$ep2$ - the last vertex in the graph

t_1 - the first term in the list of terms

t_2 - the last term in the list of terms

Output: G - a graph that satisfies all of the inputted sequences of terms

Variables:

t_0 - the current term

v_0 - the current vertex

v_{min} - the best vertex found so far

Algorithm:

```
for  $i = 1 : T.getSize()$  do
     $vmin \leftarrow NULL$ ;  $T.gotoBeginning()$ ; while
         $(t_0 \leftarrow T.nextTerm()) \neq NULL$  do
            If  $t_0.found$  Then  $continue$ ;  $v_0 \leftarrow GetBestVertex(G, t_0)$ ; if
                 $v_0.dist < vmin.dist$  then
                     $vmin \leftarrow v_0$ ;
if  $vmin \neq NULL$  then
     $t_0.found \leftarrow TRUE$ ;
    If  $adjacent(ep1, vmin)$   $AddVertices(ep1, vmin, t_1, t_0.getPrevious())$ ; if
         $t_0.getPrevious() = t_1$  then
             $AddEdge(ep1, vmin)$ ;
     $BuildGraph(G, T, ep1, vmin, t_1, t_0.getPrevious())$ ; if  $adjacent(vmin, ep2)$ 
then
         $AddVertices(vmin, ep2, t_0.getNext(), t_2)$ ;
    if  $t_0.getNext() = t_2$  then
         $AddEdge(vmin, ep2)$ ;
     $BuildGraph(G, T, vmin, ep2, t_0.getNext(), t_2)$ ;
 $AddVertices(ep1, ep2, t_1, t_2)$ ;
```

then rejected in favor of creating *vertex lists*. These vertex lists are treated as single vertices (i.e., they are atomic) except when the vertex selection algorithm wishes to select one as matching a new term. In that case, a copy of the vertex list is made which includes only the necessary terms for adjacency. In addition to acting as vertices, these vertex lists are also attached to edges to enable multiple paths between adjacent vertices. While complicated to code, it makes the creation of the production rules trivial, as each vertex corresponds to a term and each vertex list corresponds to a rule (or sequence). Once generated, the production rules are pruned to remove any rules not used by the scenario sequences.

3.8 Summary

It is time for some of the workload placed on the digital forensic laboratories to be moved back to the front line examiners. However, the only way that can happen is if the digital information is presented in a way they can understand. Digital artifacts need to be abstracted into human understandable activities for non-technical people. One way to accomplish this is by taking the digital artifacts, organizing them into simple 3-tuple terms and then creating a regular grammar from these terms. If these digital artifacts can be replaced with abstract ideas like “starting Microsoft Word”, then moving this work from digital forensic laboratories to non-technical examiners becomes feasible.

IV. Results and Analysis

The goal of this research is to determine if sequence pattern mining can be used to automate abstracting digital forensic artifacts into human understandable activities. First, the research develops new and extends existing sequence pattern mining (SPM) algorithms to work in the digital forensics domain. Next, it applies these algorithms to create terms, sequences, and production rules that abstract digital artifacts into human understandable activities. Finally, it applies these terms, sequences and production rules to a constructed dataset to test how well they generalize.

The remainder of this section begins by describing in Section 4.1 the data used. Next, Section 4.2 describes the development and implementation issues as well as design tradeoffs. In section 4.3, the development of the Use Case dataset and the use of it to develop the terms, sequences, and production rules is described. Finally, in Section 4.4, their application to the constructed dataset is discussed.

4.1 The Data

The dataset used in this research is constructed from a collection of virtual machine snapshots created in a controlled environment with Microsoft Windows 10 and Office 2013 installed to create the terms and sequences for Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Google Chrome, and Mozilla Firefox. Table 4 is a summary of the five use cases and the five to eight scenarios for each. The detailed scenarios are in Appendix A. When performing each scenario, there are several guidelines. First, after starting the virtual machine, the examiner waits 30 minutes before performing the first step of the scenario. The second is that there is at least a two minute delay between each step in the scenario. Both of these guidelines are to minimize the chances of the operating system running background processes that will complicate generating

Table 3. Extracted Data Files.

Data Type	File System Location
Event Logs	/windows/system32/winevt/logs
Registry Files	/windows/system32/config
User Data	/%USERPROFILE%/appdata/NTUSER.DAT /%USERPROFILE%/appdata/local/microsoft/ windows/UsrClass.DAT /%USERPROFILE%/appdata/local/microsoft/ windows/history/history.ie5 /%USERPROFILE%/appdata/local/google/ chrome/user data/default/History /%USERPROFILE%/appdata/roaming/mozilla/ firefox/profiles/xxxx/places.sqlite

the grammar. Once the images have been created, the files shown in Table 3 are extracted and used to create the dataset. Each scenario takes approximately 90 minutes to create and extract the data from and another 90 minutes to load into a database for use. A separate database is used for each use case and all scenarios for a single use case are in the same database.

Experiment.

The experiment takes place in two parts. The abstraction methodology described in Figure 5 is applied to the dataset to create terms, sequences, and production rules for each use case. The rules for each scenario of a use case are compared and a single consolidated rule is created. Once these generalized production rules are created, they are applied to the individual scenarios of each use case in a “leave one out” technique. For instance, the production rules are generated using scenarios 1b through 1h and then they are applied to scenario 1a to see how well the production rules generalize. While the creation of terms and sequences was also initially performed using this “leave one out” technique, the results show that they do not change appreciably.

Table 4. Use Cases.

Use Case	Scenarios for Starting Microsoft (MS) Office Applications in Windows 10
1a	User creates a MS Word document
1b	User opens MS Word from the start menu, opens an existing file from within MS Word, edits and saves it, and exits
1c	User open File Explorer, clicks on an existing file, edits and saves, it, and exits MS Word
1d	User opens Command Line, types the file name, edits and saves it, and exits MS Word
1e	User opens MS Internet Explorer and downloads and opens a document and saves it
1g	User opens Mozilla Firefox and downloads and opens a document and saves it
1h	User opens two files and copy/pastes between them
2	Same scenarios for MS Excel
3	Same scenarios for MS PowerPoint
4a	User starts Google Chrome by selecting it from the MS menu and visits a website
4b	User starts Google Chrome by selecting it from the MS menu, visits several websites and downloads/saves content
4c	User starts Google Chrome by selecting it from the MS menu, visits Facebook, logs in, and starts instant messaging
4d	User starts Google Chrome by selecting it from the MS menu, visits Twitter, logs in, and makes a tweet
4e	User starts Google Chrome by selecting it from the MS menu, visits Instagram, logs in, and views pictures
5	Same scenario for Mozilla Firefox

4.2 Database and Algorithm Design and Experiment Setup

Data Extraction.

Data extraction uses the scripts in Appendix E. While LogParser [70], The Sleuth Kit (TSK) [16], RegRipper [20] and SQLite [95] are used to extract digital artifacts, neither Log2Timeline [40] nor Plaso (a backend Python tool for Log2Timeline) [109] is. While these tools may provide access to more different types of data formats, the number of problems encountered while attempting to install them and their inconsistency in output outweighed the potential benefits they might provide. They may be implemented in the future. Instead, simple mount, copy and event log reporting

Table 5. Database Event Tables.

Timeline	start time, end time, source, activity id
Artifacts	the digital artifacts from the digital media
Loops	sequences of two or more objects
LoopSupport	cross reference between loops and timeline

Table 6. Database Production Rules.

Variables	Variables for constrained digital objects
Constraints	Constraints on digital object names, e.g., <code>^.*\/windows\/.*\$</code>
PatternVariables	cross reference between constraints and variables
Terms	A digital artifact where the object has been replaced with a variable {Actor, Action, Application, Variable}
Sequences	A sequence of two or more terms
Rules	A relation between a variable and a sequence of terms, sequences, and or rules

commands are used along with the TSK, RegRipper, LogParser, SQLite and an additional tool for extracting the NTFS Master File Tables. The only development work includes a tool for extracting the event information from the Microsoft Windows 10 operating system and a tool for parsing the extracted data into the SQLite database.

Database.

Tables 5 - 7 show the SQLite database divided into three sections. The first section is the event data, which includes the digital artifacts as well as the abstracted events created by applying the production rules. Event data includes the timeline records,

Table 7. Database Traceability.

SequenceInstances	links specific sequence with instantiation of it
SequenceCrossReference	links less abstracted events with a sequence instance
SequenceSupport	assigns variables to objects for a specific sequence instance
SequenceSupportCrossReference	cross reference for “sequence of sequences” variables

the digital artifacts, and supporting information (e.g., the digital objects, device names, URL titles, etc.). In addition, once abstraction occurs, loops are created from sequential events with the same digital object.

The second section of the database is the production rules. Consider the formal description described in Chapter III. Digital artifacts consist of two parts, the operator and the digital object(s). Each time the operator acts on a different object, even if it performs the same action, it results in a different digital artifact. Terms overcome this by replacing the specific digital object with a variable. Variables provide the capability for multiple digital artifacts to be represented with a single term by assigning different values to the variable. While variables could represent any digital object, constraints restrict the types of values that a specific variable can take. Each variable is assigned to one constraint.

Finally, the third section, traceability connections, provides traceability between the abstracted events and events at the next lower level of abstraction. As shown in Figure 4, the production rules are applied to events at a specific level of abstraction to create events at a higher abstraction level as well as traceability back to the original events and production rules.

Algorithms.

The maximum sequence size and thresholds for DVSM-V and SMTC are determined empirically. They are the largest (or smallest in the case of thresholds) values that run in a reasonable amount of time on the provided hardware. With more time and processing power, these values could be adjusted.

In the interests of speed, all data is kept in memory for as long as possible. That means existing database records are read once and kept in memory and new database records are written only once to disk. For the abstraction routines, the new database

Table 8. Use Case Dataset Compression.

	Low Level	DVSMT	SMTc
Use Case	Events	Events	Events
Use Case 1	18.6M	722K	736K
Use Case 2	16.9M	1.00M	1.9M
Use Case 3	17.1M	718K	664K
Use Case 4	17.0M	608K	582K
Use Case 5	14.6M	667K	584K

records are written to “dump” files that are later imported. Furthermore, while C++ is used, objects are only instantiated once. Rather than copying them (and, thus, creating multiple instantiations of the object), they are placed on linked lists. For instance, when a timeline record is created, the objects associated with it are instantiated. In the case of loops, this includes more than one object (e.g., hundreds) in the Object instantiation. Rather than creating a copy of this instantiation when trying to match a set of activities to a sequence, a pointer is placed on a linked list and the linked list is passed.

4.3 Data Abstraction

Data abstraction occurs in two parts. First, terms and sequences are generated for the use cases and used to abstract the digital artifacts. Next, the abstracted events that occur during the time of the human understandable user activities of interest (captured in Appendix A) are extracted and used to create production rules for each human understandable activity.

Term and Sequence Generation.

Data creation and extraction proceeds as described in Chapter III. The detailed times for each of the scenarios are shown in Appendix A. In general, each scenario takes approximately 90 minutes to create and extract the data and another 90 minutes to load into the Temporal Event Abstraction and Reconstruction (TEAR) database.

Table 9. Use Case Dataset Compression.

	DVSMT	SMTC
Use Case	Terms/Sequences	Terms/Sequences
Use Case 1	414/ 911	972/ 2,104
Use Case 2	414/ 454	941/ 540
Use Case 3	534/ 1,208	1,259/ 2,930
Use Case 4	517/ 1,077	891/ 1,563
Use Case 5	447/ 1,059	862/ 1,517

A separate database is used for each use case and all scenarios for a single use case are in the same database. As shown in Table 8, with the exception of Use Case 2, the data reduction is fairly consistent across uses cases. When SOSLA is run against use cases 1, 3, 4, and 5, the records are reduced by a little over 96%. When the events resulting from applying DVSM-V and SMTC are compared to the original dataset, the reduction is approximately the same.

Recall that each use case begins with SOSLA generating sequences that are applied to the digital artifacts. Then two instantiations of the result are created. The first uses SMTC to generate terms, sequences, and production rules which it applies to the SOSLA-abstracted events while the second uses DVSM-V for the same purpose. In the case of SMTC, this process is repeated due to hardware constraints. Each use case is then left with two sets of terms, sequences, production rules and abstracted data. As shown in Table 9, sequence and term generation varies more significantly between the use cases. Except in Use Case 2, SOSLA generates the fewest terms, followed closely by DVSM-V (observe that the DVSM-V and SMTC numbers include the terms and sequences generated by the SOSLA pre-processing). SMTC creates significantly more sequences. While increasing the threshold for sequence creation would reduce this, it would also decrease the event reduction.

Production Rule Generation.

The second step for each use case is discovering a single rule for starting the appropriate application (e.g., MS Word, MS PowerPoint, MS Excel, Chrome, Firefox). This occurs by using the detailed times in Appendix A to determine when the application begins loading and when it finishes loading. The events bookended by these times are all considered to be part of starting the application. SOSLA-Multi Object creates a single sequence for each scenario comprising all of the events within the bookends.

As an example, Table 10 shows the terms and sequences involved in starting Microsoft Word and Table 11 shows how they combine for the different scenarios. Note that there are some terms that seem to perform the Create, Modify, Access, Change on the same file twice. In reality, this is part of establishing exclusive access to the file (and is actually the file and a lock file with a similar name). Table 12 shows the production rules generated after the sequences in Table 11 are consolidated. Table 13 shows an example of the production rules used to generate the sequence for scenario UC1A. Appendix B shows the production rules and sequences generated for each of the use cases. There are three cases where a specific scenario for a specific use case produced too few (three - five) terms and sequences that it was excluded from the production rule generation. In all cases, when the generated production rules are applied to the use case datasets, the events used to generate the rules are found and no other sequences of events are found.

4.4 Results

Once the terms and sequences are generated from the Use Case dataset, they are then tested using a “leave one out” technique to see how well they generalize to new data. Each use case is composed of between five and eight scenarios. For each use

case, N tests are done where N is the number of scenarios. For each test, one scenario is left out when the production rules are generated. Then the production rules are tested on that scenario to see if the actual user activity (e.g., starting Microsoft Word) is found and if the production rules “find” any other instances of the user activity. These are called *true positives* and *false positives* respectively. If the true positive is missed, this is called a *false negative*. The similarity threshold is the maximum number of edit operations needed using the Levenshtein edit distance to transform one sequence into the other. The threshold was set empirically to ensure that there are no false negatives.

As can be seen from the results, the similarity threshold was consistent at between 2 and 4 edit operations. This threshold enabled all user activities to be identified while keeping the false positives low. To put the number of false positives in context, each scenario had over 100,000 events. That means that the number of false positives could be as high as 100,000 (assuming each event is the first event in exactly one sequence). Thus, 30 false positives is equivalent to a false positive rate of no more than 0.03%.

While the experiment shows the method is successful at generalizing to unseen data to find Microsoft Office applications starting, it is not successful at generalizing to unseen instances of web browsers starting. When the method is applied to use case 4 and 5, it failed. The first issue that arose was identifying when the “start application” process completed. The methodology calls for recording when each step in each scenario starts, but not when it ends. The assumption was that when it was complete, there would be a period of time when no digital artifacts occurred. That was not the case. While repeating the experiments where the start and end times are explicitly recorded may solve this problem, there is reason to believe it will not. Some processes continue to perform actions long after the user interface is completely loaded. In essence, the user observes that the application start up is complete when

it isn't. If so, the only way to identify these applications starting is with expert knowledge.

4.5 Summary

The goal of this research is to determine if sequence pattern mining is effective at automating the abstraction of digital artifacts into human understandable activities. The three algorithms, SOSLA, DVSM-V, and SMTC all compress the temporal data by 96%. The production rules generated successfully find the human understandable activities with a minimal number of false positives. The highest number of false positives for a specific scenario was 29 which is sufficiently low to allow a digital forensic examiner to perform their work efficiently. 29 false positives results in a maximum false positive rate of 0.03%. However, the average false positive rate is 0.0079% for SMTC and 0.0104% for DVSM-V. While setting the similarity threshold to guarantee all positive instances are found prevents us from knowing how well the algorithm generalizes to previously unseen data, the fact that in all cases, a threshold of 4 is sufficient suggests that a similar threshold will likely work for new datasets.

Table 10. Representative Terms and Sequences.

Term/Sequence	Description
T13	Modify File :*\$
T112	System Modify Registry Key :*\$
T150	User Modify Registry Key :*\$
T205	Change File :*\$
T145	System Modify Registry Key :*/Windows/.*\$
T230	User Modify Registry Key :*/Windows/.*\$
T310	User Access File :*\$
S347	Modify and Change File :*/Windows/.*/*Windows/.*\$
S667	Modify and Change File :*/Windows/.*\$
S676	Modify, Access and Change File :*/Windows/.*\$
S677	Create, Modify, Access, Change, Create, Modify, Access, and Change File :*/Windows/.*\$
S706	Modify, Access and Change File :*\$
S757	Create, Modify, Access, Change, Create, Modify, Access, and Change File :*/Users/user/AppData/.*/*Microsoft/.*\$
S754	Create, Modify, Access, Change, Create, Modify, Access, and Change File :*/Users/user/AppData/Local/.*\$
S759	Modify and Change File :*/Users/user/AppData/.*/*Microsoft/.*\$
S766	Modify, Access, and Change File :*/Users/user/AppData/.*/*Microsoft/.*\$
S768	Create :*/Users/user/AppData/.*/*Microsoft/.*\$
S777	Modify, Access, Change, Modify, Access and Change File :*/Users/user/AppData/.*/*Microsoft/.*\$
S940	Modify, Access, and Change File :*/Users/user/AppData/Local/.*\$
S1080	Create, Modify, Access, Change, Create, and Access File :*/Users/user/AppData/.*/*Microsoft/.*\$

Table 11. Example Sequences for Starting Microsoft Word.

Scenario	Sequence
UC1A	667 230 112 150 1080 766 757 667 112 667 667 676 677 667 230 145 667
UC1B	667 667 667 310 230 667 230 112 150 145 667 112 667 667 112 676 677 667 145
UC1C	230 667 112 667 667 667 150 230 112 145 667 667 676 677 667 667 112
UC1D	13 230 706 768 205 150 230 667 150 230 667 112 145 667 676 677 667 667 112 766 777 759 766 759 777 230 150 112 766 757 759 310 230 230 759 759 150 230 150 766 150 112 145 940 150
UC1E	768 205 230 766 757 230 667 766 777 759 150 667 112 145 667 676 677 667 754 667 150 766 759 777 150 230
UC1F	150 766 759 766 676 677 1070 150 836 214 150 145 112 428 735 940 228 667 735 1196 735 214 150 667 347
UC1G	667 230 112 150 112 667 150 145 676 677 145 667 706 145 1070 112 667 667
UC1H	940 377 228 667 112 150 759 150 112 667 643 676 1050 667 150 667 1063 112 40 766 315 490 202 112 667 150 230 145 766 757 667 230 667 490 48 759 667 13 74 8 40 667 342 677 757 766 230 667 112 677 667 230 754 667 230 777 667 112 112 676 686 112 1047 112 676 667 202 667 706 145 145 766 777 759 694 150 667 759 145 706 150 777 150 759 230 112 766 759 777 230 112 754 766 940 766 150 377 667

Table 12. Representative Production Rules.

ID	Rule
S	T0 + S1010 + T0
S3	T112 + T150 + T1080 + T766 + T757
S12	T676 + T677
S21	T112 + T667
S28	T230 + T145
S84	S21 + T667 + S12
S252	T667 + T230 + S3
S284	S252 + T667 + S84
S285	S284 + T667 + S28
S1010 (b)	S285 + S1012
S1012 (f)	T667

Table 13. Example of Production Rules Generating a Sequence.

S1010 (b)
S285 + S1012 (f)
S284 + T667 + S28 + S1012 (f)
S252 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + S3 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + S84 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + S21 + T667 + S12 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + S12 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + S28 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + T230 + T145 + S1012 (f)
T667 + T230 + T112 + T150 + T1080 + T766 + T757 + T667 + T112 + T667 + T667 + T676 + T677 + T667 + T230 + T145 + T667

Table 14. Production Rule Testing for SMTC.

Use Case Scenario	Similarity Threshold	True Positives	False Positives	False Negatives
UC1A	3	1	1	0
UC1B	3	1	1	0
UC1C	3	1	0	0
UC1D	3	1	2	0
UC1E	3	1	1	0
UC1F	3	1	1	0
UC1G	3	1	0	0
UC1H	3	1	0	0
UC2A	3	1	0	0
UC2B	3	1	0	0
UC2C	2	1	0	0
UC2E	3	1	0	0
UC2F	3	1	1	0
UC2H	4	1	21	
UC3A	4	1	17	0
UC3B	4	1	15	0
UC3D	4	1	18	0
UC3E	4	1	25	0
UC3F	4	1	21	0
UC3G	4	1	20	0
UC3H	4	1	22	0

Table 15. Production Rule Testing for DVSMT.

Use Case Scenario	Similarity Threshold	True Positives	False Positives	False Negatives
UC1A	3	1	1	0
UC1B	3	1	0	0
UC1C	3	1	1	0
UC1D	3	1	2	0
UC1E	3	1	2	0
UC1F	3	1	0	0
UC1G	3	1	0	0
UC1H	3	1	0	0
UC2A	4	1	9	0
UC2B	4	1	5	0
UC2C	2	1	3	0
UC2E	4	1	24	0
UC2F	3	1	8	0
UC2H	3	1	16	0
UC3A	3	1	29	0
UC3B	3	1	27	0
UC3D	4	1	26	0
UC3E	2	1	19	0
UC3F	2	1	19	0
UC3G	2	1	9	0
UC3H	3	1	20	0

V. Conclusions

Digital Forensic Laboratories are being overwhelmed with new and bigger demands. The goal of this research is to determine if sequence pattern mining can be used to automate abstracting digital artifacts into human understandable activities, resulting in a large time savings for the laboratories. One existing sequence pattern mining (SPM) algorithm, Discontinuous Varied Order Sequence Mining (DVSM), is extended to work in the digital forensics domain (DVSM with Variables) and two new SPM algorithms, Single Object Sequence and Loop Abstraction (SOSLA) and Sequence Mining of Temporal Clusters (SMTTC), are created. These algorithms extend SPM by attaching attribute variables to items and by developing a new method for addressing interleaving. First, the algorithms are used to extract terms and sequences from a dataset and then those sequences are consolidated to create production rules for starting user specified applications. A dataset is constructed with use cases that demonstrate different human activities including starting Microsoft Word, PowerPoint, and Excel and starting Mozilla Firefox and Google Chrome. When the algorithms are applied to the dataset, the resulting terms and sequences compress the data by 96%. When the production rules are generated and applied to the use cases using a “leave one out” approach, all true positives are found while the false positive rate stays below 0.03%.

5.1 Hypothesis

This research tests the hypothesis that sequence pattern mining algorithms can automate discovery of terms, sequences, and production rules and use them to reduce long sequences of digital artifacts into single labels of human activities. Specifically, the research is evaluated to see if the sequence pattern mining algorithms (SPMs)

“provide an automated method for generating a set of sequences of production rules that enable describing with a single rule” the human activities listed above and measuring how successful the sequences and production rules are at identifying occurrences of the activities.

5.2 Algorithms

The three SPMs developed in this research are Apriori algorithms, i.e., a class that extend the first SPM developed by Agrawal [2]. All three include the capability of attaching variables to items. For instance, rather than having one item for making a roast beef sandwich and a second for making a turkey sandwich, there is one item, make sandwich, with a variable that describes the contents. Single Object Sequence and Loop Abstraction (SOSLA) is a greedy SPM that only creates sequences where every item has the same value for the variable. It performs well at creating human understandable sequences, but its greedy nature makes it extending it to multiple variable assignments ineffective. Discontinuous Varied Order of Sequence Mining with Variable (DVSM-V) takes the existing DVSM algorithm [84] and adds variables to it. It begins by creating sequences of length 1, removing any it doesn’t consider “interesting” and then creating sequences of length 2 by taking the remaining sequences and adding the items that immediately precede or succeed them. The process is repeated either a fixed number of iterations or until no new sequences are added. Finally, Sequence Mining of Temporal Clusters begins by creating sets of temporally close items, removing any sets it doesn’t consider “interesting” and then choosing a representative sequence from each set.

Observe that choosing a method for determining whether a sequence is “interesting” and choosing a method for identifying a representative sequence from a set of temporally close items is SPM-independent. For this research, DVSM-V uses the

same measurement that DVSM [84] uses,

$$\frac{DL(D)}{DL(a) + DL(D|a)} \quad (3)$$

where D is the dataset, a is a potential sequence and DL is the number of bytes to encode the argument) while SMTC uses a new measurement,

$$occurrences(C) * |C|^{variableCount(C)*\epsilon} \quad (4)$$

$Occurrences(C)$ is the number of times this cluster of terms occurs in the dataset. $VariableCount(C)$ is the number of distinct variables in C , and ϵ is a user specified weighting. To determine a representative sequence, SMTC chooses the median of the set (i.e., the sequence that is closest to every other sequence in the set). By first performing clustering of temporally close items and creating the power set of those items, SMTC specifically addresses the problem of interleaving.

5.3 Experimental Methodology

The experiment begins by constructing a dataset that demonstrates the user activities. Each activity is considered a use case and between five and eight scenarios are created for each. The scenarios are enacted in a virtual machine with the start time of each step recorded. Once the dataset is constructed, the digital artifacts are extracted and placed in separate SQLite databases for each use case. Then, the SPM algorithms are used to create terms (items with variables) and sequences. Next, the datasets are compressed using the terms and sequences. Finally, production rules are created and tested using a “leave one out” method. For each use case, N tests are performed where N is the number of scenarios. For each test, one scenario is left out and the production rules are generated using the remainder. Then, the production

rules are tested on the left out scenario to see how well the rules generalize.

5.4 Results

The algorithms are successful at developing a set of production rules for describing a user activity with a single (top level) rule. The SPM algorithms reduce the data by 96%. When the “leave one out” testing is performed, the true positives are discovered while the false positive rate is kept below 0.03%.

5.5 Discussion

While the research is successful, several concerns emerged during testing. First, the use cases were created to minimize interleaving. The analyst waited a period of time between each step to ensure that the previous step was completed and that no other processing was occurring. Furthermore, when interleaving did occur, the production rule generation algorithm ignored terms that were manually identified as applying to another activity. This resulted in SMTC performing on par with DVSM-V. SMTC’s potentially better performance at avoiding interleaving was not tested.

The second concern is that the threshold for similarity was set to ensure that all tests found the actual human activity (i.e., each test had true positives but no false negatives). This was done to measure whether the false positive rate was low enough to be acceptable. While it is, it is unclear whether setting the threshold ahead of time would have resulted in false negatives. However, given the low variability in the threshold (i.e., between 2 and 4 edit operations across all tests), setting a threshold of 4 edit operations would have resulted in no false negatives while likely only raising the false positive rate to the levels of the other tests.

5.6 Next Steps

The current results only show the algorithms applied to the use cases for starting the applications. Subsequent tests should be run using the existing data for performing simple tasks in the applications. Once all of these user activities are modeled as well as some system processes (e.g., automated software updates), a technical narrative of a “day in the life” can be presented. Further use cases would provide additional rules for better results when applied to unseen datasets.

5.7 Impact

The SOSLA, DVSM-V, and SMTC algorithms are the first attempt at attaching variables to Apriori sequence pattern mining (SPM) algorithms in a domain with a large number of items. Their use has improved SPM in the digital forensics domain. Applying these algorithms to a Use Case dataset has resulted in production rules for recognizing user activities and significantly reducing the amount of information analysts need to sift through.

Appendix A. Scenarios

All times are in Pacific Daylight Time

1.1 Use Case 1 - Microsoft (MS) Word in Windows 10

User Creates a MS Word Document

29-May-2017 11:32 Start MS Windows
29-May-2017 12:03 Open MS Word from Start Menu
29-May-2017 12:06 Create and save File1 (under 100 words)
29-May-2017 12:08 Close File1
29-May-2017 12:11 Shut down MS Windows

User Starts MS Word from the Start Menu and Edits an Existing Document

29-May-2017 18:37 Start MS Windows
29-May-2017 19:02 Open File Explorer. Navigate to My Documents and double-click on File1
29-May-2017 19:07 Edit and save File1
29-May-2017 19:10 Close File1
29-May-2017 20:16 Shut down MS Windows

User Starts MS Word from the Command Line and Edits an Existing Document

29-May-2017 20:33 Start MS Windows
29-May-2017 21:03 Open cmd.exe. Change Directory to My Documents and enter File1
29-May-2017 21:06 Edit and save File1
29-May-2017 21:09 Exit MS Word and cmd.exe
29-May-2017 21:11 Shut down MS Windows

User Starts MS Edge, Downloads a Document and Views It

31-May-2017 18:36 Start MS Windows
31-May-2017 19:17 Start MS Edge
31-May-2017 19:53 Visit www.boardgamegeek.com.
Search on Snowball Fighting and
open rules.doc
31-May-2017 19:55 Close rules.doc, MS Word and MS
Edge
31-May-2017 19:58 Shut down MS Windows

User Starts Google Chrome, Downloads a Document and Views It

1-Jun-2017 11:47 Start MS Windows
1-Jun-2017 12:19 Start Google Chrome
1-Jun-2017 12:28 Visit www.boardgamegeek.com.
Search on Snowball Fighting and
open rules.doc
1-Jun-2017 12:31 Close rules.doc, MS Word and
Google Chrome
1-Jun-2017 12:32 Shut down MS Windows

User Starts Mozilla Firefox, Downloads a Document and Views It

1-Jun-2017 16:20 Start MS Windows
1-Jun-2017 16:49 Start Mozilla Firefox
1-Jun-2017 16:53 Visit www.boardgamegeek.com.
Search on Snowball Fighting and
open rules.doc
1-Jun-2017 16:56 Close rules.doc, MS Word and
Mozilla Firefox
1-Jun-2017 16:57 Shut down MS Windows

User Starts MS Word from the Start Menu, Creates and Saves a Document

2-Jun-2017 11:14 Start MS Windows
2-Jun-2017 11:47 Start MS Word from the Start Menu.
Create and edit File2
– under 100 words
2-Jun-2017 11:53 Save File2
2-Jun-2017 11:57 Close MS Word
2-Jun-2017 11:59 Shut down MS Windows

**User Starts MS Word from the Start Menu, Opens two documents,
Copies and Pastes part of one document to the other**

3-Jun-2017 09:28 Start MS Windows
3-Jun-2017 09:58 Start MS Word from the Start Menu
3-Jun-2017 10:10 Open File2
3-Jun-2017 10:23 Open File1
3-Jun-2017 10:30 Copy and Paste a section of File1 to
File2
3-Jun-2017 10:39 Save File2
3-Jun-2017 10:44 Close MS Word
3-Jun-2017 10:49 Shut down MS Windows

1.2 Use Case 2 - Microsoft (MS) PowerPoint in Windows 10

User Creates a MS PowerPoint Presentation

3-Jun-2017 14:19 Start MS Windows
3-Jun-2017 14:41 Open MS PowerPoint from Start Menu
3-Jun-2017 14:51 Create and save Power1 – one slide
3-Jun-2017 14:59 Close Power1
3-Jun-2017 15:59 Shut down MS Windows

User Starts MS PowerPoint the Start Menu and Edits an Existing Presentation

5-Jun-2017 11:55 Start MS Windows
5-Jun-2017 12:38 Start MS PowerPoint from the Start Menu
5-Jun-2017 12:48 Open Power1 and start editing it – add second slide
5-Jun-2017 12:53 Save Power1
5-Jun-2017 13:00 Close MS PowerPoint
5-Jun-2017 13:05 Shut down MS Windows

User Starts MS PowerPoint from File Explorer and Edits an Existing Presentation

5-Jun-2017 14:10 Start MS Windows
5-Jun-2017 14:44 Open File Explorer from the Start Menu. Navigate to My Documents and click on Power1
5-Jun-2017 14:50 Edit and save Power1
5-Jun-2017 14:54 Save Power1 a second time
5-Jun-2017 14:57 Exit MS PowerPoint and close File Explorer
5-Jun-2017 14:59 Shut down MS Windows

User Starts MS PowerPoint from the Command Line and Edits an Existing Presentation

5-Jun-2017 15:36 Start MS Windows
5-Jun-2017 16:07 Start cmd.exe from the Start Menu
5-Jun-2017 16:08 Change Directory to My Documents
and enter Power1.ppt
5-Jun-2017 16:12 Edit document and save it – third
slide
5-Jun-2017 16:13 Close Power1, MS PowerPoint and
cmd.exe
5-Jun-2017 16:15 Shut down MS Windows

User Starts Google Chrome, Downloads a Presentation and Views It

6-Jun-2017 11:05 Start Google Chrome
6-Jun-2017 11:12 Visit www.boardgamegeek.com.
Searched on Sail to India
6-Jun-2017 11:18 Download and open PlayerAid.ppt
6-Jun-2017 12:23 Close PlayerAid.ppt, MS PowerPoint
and Google Chrome
6-Jun-2017 12:28 Shut down MS Windows

User Starts MS Edge, Downloads a Presentation and Views It

6-Jun-2017 13:41 Start MS Windows
6-Jun-2017 14:14 Start MS Edge
6-Jun-2017 14:17 Visit www.boardgamegeek.com.
Search on Sail to India
6-Jun-2017 14:21 Download and open PlayerAid.ppt
6-Jun-2017 14:25 Close PlayerAid.ppt, MS PowerPoint
and MS Edge
6-Jun-2017 14:28 Shut down MS Windows

User Starts Mozilla Firefox, Downloads a Presentation and Views It

6-Jun-2017 15:06 Start MS Windows
6-Jun-2017 15:37 Start Mozilla Firefox
6-Jun-2017 15:39 Visit www.boardgamegeek.com.
Search on Sail to India
6-Jun-2017 15:46 Download and open PlayerAid.ppt
6-Jun-2017 15:48 Close PlayerAid.ppt, MS PowerPoint
and Mozilla Firefox
6-Jun-2017 15:50 Shut down MS Windows

User Starts MS PowerPoint from the Start Menu, Opens Two Presentations, Copies and Pastes part of One Presentation to the Other

6-Jun-2017 21:55 Start MS Windows
6-Jun-2017 22:27 Start MS PowerPoint from the Start
Menu
6-Jun-2017 22:33 Open Power1.ppt and PlayerAid.ppt
6-Jun-2017 22:39 Copy and Paste a section of PlayerAid.ppt to Power1.ppt
6-Jun-2017 22:39 Save Power1.ppt
6-Jun-2017 22:44 Close MS PowerPoint
6-Jun-2017 22:52 Shut down MS Windows

1.3 Use Case 3 - Microsoft (MS) Excel in Windows 10

User Creates a MS Excel Spreadsheet

14-Jun-2017 8:07 Start MS Windows
14-Jun-2017 8:53 Open MS Excel from Start Menu
14-Jun-2017 9:01 Create Spread1.xls – 50 cells
14-Jun-2017 9:05 Save Spread1.xls
14-Jun-2017 9:08 Close Spread1.xls
14-Jun-2017 9:10 Shut down MS Windows

User Starts MS Excel the Start Menu and Edits an Existing Spreadsheet

14-Jun-2017 10:15 Start MS Windows
14-Jun-2017 10:48 Start MS Excel from the Start Menu
14-Jun-2017 10:50 Open Spread1.xls and start editing it
– add summations
14-Jun-2017 10:52 Save Spread1.xls
14-Jun-2017 10:54 Close MS Excel
14-Jun-2017 10:56 Shut down MS Windows

User Starts MS Excel from File Explorer and Edits an Existing Spreadsheet

14-Jun-2017 12:17 Start MS Windows
14-Jun-2017 12:49 Open File Explorer from the Start
Menu. Navigate to My Documents
and click on Spread1.xls
14-Jun-2017 12:53 Edit and save Spread1.xls
14-Jun-2017 12:57 Exit MS Excel and close File Ex-
plorer
14-Jun-2017 12:59 Shut down MS Windows

User Starts MS Excel from the Command Line and Edits an Existing Spreadsheet

14-Jun-2017 14:18 Start MS Windows
14-Jun-2017 14:51 Start cmd.exe from the Start Menu.
Change Directory to My Documents
and enter Spread1.xls
14-Jun-2017 14:58 Tried a second time and this time
spread1.xls opened
14-Jun-2017 15:01 Edit Spread1.xls – more summation
14-Jun-2017 15:03 Close Spread1, MS Excel and
cmd.exe
14-Jun-2017 15:06 Shut down MS Windows

User Starts Google Chrome, Downloads a Spreadsheet and Views It

14-Jun-2017 15:59 Start MS Windows
14-Jun-2017 20:10 Start Google Chrome. Visit
www.boardgamegeek.com. Search
on Star Wars Destiny
14-Jun-2017 20:19 Download and open ChardCheck-
List.xlsx
14-Jun-2017 20:21 Save CardCheckList.xlsx
14-Jun-2017 20:24 Close MS Excel and Google Chrome
14-Jun-2017 20:26 Shut down MS Windows

User Starts MS Edge, Downloads a Spreadsheet and Views It

14-Jun-2017 22:21 Start MS Windows
14-Jun-2017 23:02 Start MS Edge
14-Jun-2017 23:04 Visit www.boardgamegeek.com.
Search on Star Wars Destiny
14-Jun-2017 23:07 Download and open CardCheck-
List.xlsx
14-Jun-2017 23:10 Save CardCheckList.xlsx
14-Jun-2017 23:11 Close MS Excel and MS Edge
14-Jun-2017 23:12 Shut down MS Windows

User Starts Mozilla Firefox, Downloads a Spreadsheet and Views It

15-Jun-2017 8:30 Start MS Windows
15-Jun-2017 9:05 Start Mozilla Firefox
15-Jun-2017 9:08 Visit www.boardgamegeek.com.
Search on Star Wars Destiny
15-Jun-2017 9:16 Download and open CardCheck-
List.xlsx
15-Jun-2017 9:18 Save CardCheckList.xlsx
15-Jun-2017 9:20 Close MS Excel and Mozilla Firefox
15-Jun-2017 9:24 Shut down MS Windows

**User Starts MS Excel from the Start Menu, Opens Two Spreadsheets,
Copies and Pastes Part of One Spreadsheet to the Other**

15-Jun-2017 11:16 Start MS Excel from the Start Menu
15-Jun-2017 11:20 Open Spread1.xls and CardCheck-
List.xlsx
15-Jun-2017 11:26 Copy and Paste a section of Card-
CheckList.xlsx to Spread1.xlsx
15-Jun-2017 11:28 Save Spread1.xls
15-Jun-2017 11:29 Close MS Excel
15-Jun-2017 11:30 Shut down MS Windows

1.4 Use Case 4 - Google Chrome in Windows 10

User Starts Google Chrome and Visits Some Websites

22-Jun-2017 20:02 Start Microsoft Windows
22-Jun-2017 21:02 Start Google Chrome
22-Jun-2017 21:04 Visit www.nytimes.com
22-Jun-2017 21:08 Close Google Chrome
22-Jun-2017 21:10 Shut down Microsoft Windows

User Starts Google Chrome, Visit a Website and Download Content

22-Jun-2017 22:04 Start Microsoft Windows
22-Jun-2017 22:47 Start Google Chrome. Search on
sunset images. Click on one.
22-Jun-2017 22:51 Right-click on image and save it to
the Download directory
22-Jun-2017 22:57 Close Google Chrome
22-Jun-2017 23:00 Shut down Microsoft Windows

User Starts Google Chrome, Visit a Website and Downloads Content

28-Jun-2017 12:52 Start Microsoft Windows
28-Jun-2017 13:22 Start Google Chrome
28-Jun-2017 13:30 Visit www.boardgamegeek.com
28-Jun-2017 13:32 Navigate to Snowball Fighting
28-Jun-2017 13:34 Download rules.doc
28-Jun-2017 13:37 Close Google Chrome
28-Jun-2017 13:45 Shut down Microsoft windows

User Starts Google Chrome, Visits Facebook and Instant Messages

23-Jun-2017 12:07 Start Google Chrome
23-Jun-2017 12:12 Visit www.facebook.com
23-Jun-2017 12:47 Instant Message Amanda
23-Jun-2017 13:17 Close Google Chrome
23-Jun-2017 13:20 Shut down Microsoft windows

User Starts Google Chrome, Visits Twitter and Tweets

23-Jun-2017 14:23 Start Microsoft Windows
23-Jun-2017 14:53 Start Google Chrome
23-Jun-2017 14:57 Visit www.twitter.com
23-Jun-2017 14:59 Tweet
23-Jun-2017 15:02 Close Google Chrome
23-Jun-2017 15:05 Shut down Microsoft windows

User Starts Google Chrome, Visits Instagram and Comments on an Image

23-Jun-2017 18:57 Start Google Chrome
23-Jun-2017 19:02 Visit www.instagram.com
23-Jun-2017 19:03 Log in
23-Jun-2017 19:08 Search on sunset and select one
23-Jun-2017 19:09 Comment on the image
23-Jun-2017 19:12 Log out of Instagram
23-Jun-2017 19:14 Close Google Chrome
23-Jun-2017 19:16 Shut down Microsoft windows

1.5 Use Case 5 - Mozilla Firefox in Windows 10

User Starts Mozilla Firefox and Visits Some Websites

26-Jun-2017 11:08 Start Mozilla Firefox
26-Jun-2017 11:14 Visit www.nytimes.com
26-Jun-2017 11:16 Close Mozilla Firefox
26-Jun-2017 11:18 Shut down Microsoft Windows

User Starts Mozilla Firefox, Visits a Website and Downloads Content

26-Jun-2017 12:40 Start Microsoft Windows
26-Jun-2017 13:19 Start Mozilla Firefox
26-Jun-2017 13:30 Search on sunset images. Click on
one
26-Jun-2017 13:41 Right-click on image and save it to
the Download directory
26-Jun-2017 13:48 Close Mozilla Firefox
26-Jun-2017 13:51 Shut down Microsoft Windows

User Starts Mozilla Firefox, Visits a Website and Downloads Content

26-Jun-2017 19:26 Start Microsoft Windows
26-Jun-2017 19:41 Start Mozilla Firefox
26-Jun-2017 19:46 Visit www.boardgamegeek.com and
search on Snowball Fighting
26-Jun-2017 19:50 Download rules.doc
26-Jun-2017 19:53 Close Mozilla Firefox
26-Jun-2017 19:57 Shut down Microsoft windows

User Starts Mozilla Firefox, Visits Facebook and Instant Messages

26-Jun-2017 22:37 Start Mozilla Firefox
26-Jun-2017 23:01 Visit www.facebook.com
26-Jun-2017 23:03 Instant Message Amanda
26-Jun-2017 23:05 Close Mozilla Firefox
26-Jun-2017 23:20 Shut down Microsoft windows

User Starts Mozilla Firefox, Visits Twitter and Tweets

27-Jun-2017 8:12 Start Mozilla Firefox
27-Jun-2017 8:17 Visit www.twitter.com and log in
27-Jun-2017 8:19 Tweet
27-Jun-2017 8:21 Close Mozilla Firefox
27-Jun-2017 8:37 Shut down Microsoft windows

User Starts Mozilla Firefox, Visits Instagram and Comments on an Image

27-Jun-2017 11:12 Start Microsoft Windows
27-Jun-2017 12:08 Start Mozilla Firefox
27-Jun-2017 12:16 Visit www.instagram.com and log in
27-Jun-2017 12:18 Search on sunset and select one
27-Jun-2017 12:20 Comment on the image
27-Jun-2017 12:22 Close Mozilla Firefox
27-Jun-2017 12:24 Shut down Microsoft windows

Appendix B. Production Rules for Starting Applications

2.1 Use Case 1 - Microsoft (MS) Word in Windows 10

Use Case 1 is centered around different ways of starting Microsoft (MS) Word. The first scenario (A) is starting MS Word from the Microsoft Menu and creating a document while the next three scenarios (B - D) are modifying that same document but starting MS Word from the Microsoft Menu, File, Explorer, and the Command Line respectively. The next three scenarios (E - G) open a web browser (MS Edge, Google Chrome and Mozilla Firefox respectively) and then open MS Word by clicking on an appropriate document in the browser. Finally, the last scenario (H) opens MS Word from the Microsoft Menu, opens two documents and cut and pastes information between them.

For Use Case 1, the term lists are identical between the DVSMT and SMTC datasets. This indicates that they are composed exclusively of terms and single object strings (i.e., strings generated by SOSLA and not by DVSMT or SMTC). As a result, there is only one set of tables for the two datasets.

Term/String	Use Case 1 Description
T8	Access File ^.*\$
T13	Modify File ^.*\$
T40	Access File ^.*\Windows\.*\$
T112	System Modify Registry Key ^.*\$
T150	User Modify Registry Key ^.*\$
T202	Change Metainfo File ^.*\windows\.*\$
T205	Change Metainfo File ^.*\$
T145	System Modify Registry Key ^.*\Windows\.*\$
T228	Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
T230	User Modify Registry Key ^.*\Windows\.*\$
T310	User Access File ^.*\$
T315	Change Metainfo File ^.*\\$OrphanFiles\.*\$
S342	Create, Modify, Access and Change Metainfo File ^.*\\$OrphanFiles\.*\$
S347	Modify and Change Metainfo File ^.*\Windows\.*\Windows\.*\$
S377	Modify and Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
S490	Modify and Change Metainfo File ^.*\Windows\Logs\.*\$
S643	Create, Access, Create, Access ^.*\Windows\.*\$
S667	Modify and Change Metainfo File ^.*\Windows\.*\$
S676	Modify, Access and Change Metainfo File ^.*\Windows\.*\$
S677	Create, Modify, Access, Change Metainfo, Create, Modify, Access, and Change Metainfo File ^.*\Windows\.*\$

Term/String	Use Case 1 Description
S686	Create, Modify Access and Change Metainfo File ^.*\Windows\Logs\.*\$
S694	Modify, Access, Change Metainfo, Modify, Access, and Change Metainfo File ^.*\$
S706	Modify, Access and Change Metainfo File ^.*\$
S754	Create, Modify, Access, Change Metainfo, Create, Modify, Access, and Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
S757	Create, Modify, Access, Change Metainfo, Create, Modify, Access, and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S759	Modify and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S766	Modify, Access, and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S768	Create ^.*\Users\user\AppData\.*\Microsoft\.*\$
S777	Modify, Access, Change Metainfo, Modify, Access and Change MetaInfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S940	Modify, Access, and Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
S1047	Modify, Access, and Change Metainfo File ^.*\Windows\Logs\.*\$

Term/String	Use Case 1 Description
S1050	Modify, Change Metainfo, Modify, Change Metainfo ^.*\Windows\.*\$
S1063	Create, Modify, Access, Change Metainfo, Create, and Access File ^.*\\$OrphanFiles\.*\$
S1080	Create, Modify, Access, Change, Create, and Access File ^.*\Users\user\AppData\.*\Microsoft\.*\$

Scenario	Sequence of Terms and Strings for Starting Microsoft Word
UC1A	667 230 112 150 1080 766 757 667 112 667 667 676 677 667 230 145 667
UC1B	667 667 667 310 230 667 230 112 150 145 667 112 667 667 112 676 677 667 145
UC1C	230 667 112 667 667 667 150 230 112 145 667 667 676 677 667 667 112
UC1D	13 230 706 768 205 150 230 667 150 230 667 112 145 667 676 677 667 667 112 766 777 759 766 759 777 230 150 112 766 757 759 310 230 230 759 759 150 230 150 766 150 112 145 940 150
UC1E	768 205 230 766 757 230 667 766 777 759 150 667 112 145 667 676 677 667 754 667 150 766 759 777 150 230
UC1F	150 766 759 766 676 677 150 150 145 112 940 228 667 150 667 347
UC1G	667 230 112 150 112 667 150 145 676 677 145 667 706 145 112 667 667
UC1H	940 377 228 667 112 150 759 150 112 667 643 676 1050 667 150 667 1063 112 40 766 315 490 202 112 667 150 230 145 766 757 667 230 667 490 48 759 667 13 74 8 40 667 342 677 757 766 230 667 112 677 667 230 754 667 230 777 667 112 112 676 686 112 1047 112 676 667 202 667 706 145 145 766 777 759 694 150 667 759 145 706 150 777 150 759 230 112 766 759 777 230 112 754 766 940 766 150 377 667

ID	Use Case 1 Production Rule
S1	T667 T667 T310
S2	T667 S1
S3	T112 T150 T1080 T766 T757
S4	T667
S12	T676 T677
S15	T112 S4 T150 T145 T676 T677 T145
S16	T667 T230 T112 T150 S15
S21	T112 T667
S28	T230 T145
S30	T112 T667
S33	S16 T667 T706 T145 S30
S34	S33
S42	T667 T112
S45	T667 T112 T766 T777 T759 T766 T759 T777 T230 T150 T112 T766 T757 T759 T310 T230 T230 T759 T759 T150 T230 T150 T766 T150 T112 T145 T940 T150
S48	T112 T145 T667 T676 T677 T667 T754 T667 T150 T766 T759 T777 T150 T230
S60	T112 T676 T677
S141	T150 T145
S142	T230 T112 S141
S191	S142 T667 S21
S239	S21 T667 S12

ID	Use Case 1 Production Rule
S433	T150 T230
S532	T145
S533	T112 S532 T667
S534	S433 S533
S582	S433 T667 T112 T145
S631	S2 T230
S773	S631 T667 S191
S789	S773 T667 S60
S792	S789 T667 T145
S1187	T667 T230 S3
S1375	S1187 T667 S239
S1376	S1375 T667 S28
S1389	S1376 T667
S2299	T112
S3414	T230 T667 S2299 T667 T667
S3880	S3414 T667 S534
S3911	S3880 T667 S12
S3916	S3911 T667 S42
S4530	T230 T706 T768 T205 S433
S4531	T13 S4530
S5041	S4531 T667 S582
S5072	S5041 T667 S12
S5079	S5072 T667 S45

ID	Use Case 1 Production Rule
S5087	T766 T757
S5088	T205 T230 S5087
S5089	T768 S5088
S5694	T230 T667 T766 T777 T759 T150
S5697	S5089 S5694
S5707	S5697 T667 S48
S5719	T150 T766 T759 T766 T676 T677 T150 T150 T145
S5727	T112 T940 T228
S5730	S5719 S5727
S5748	S5730 T667 T150
S5760	S5748 T667 T347
S5764	S34 T667
S5770	T940 T377 T228 T667
S5776	T112 T150 T759 T150 T112
S5782	T676 T1050
S5783	S5776 T667 T643 S5782
S5786	S5770 S5783
S5792	S5786 T667 T150

ID	Use Case 1 Production Rule
S5799	S5792 T667 T1063 T112 T40 T766 T315 T490 T202 T112 T667 T150 T230 T145 T766 T757 T667 T230 T667 T490 T48 T759 T667 T13 T74 T8 T40 T667 T342 T677 T757 T766 T230 T667 T112 T677 T667 T230 T754 T667 T230 T777 T667 T112 T112 T676 T686 T112 T1047 T112 T676 T667 T202 T667 T706 T145 T145 T766 T777 T759 T694 T150 T667 T759 T145 T706 T150 T777 T150 T759 T230 T112 T766 T759 T777 T230 T112 T754 T766 T940 T766 T150 T377 T667
S5802 (a)	S792
S5802 (b)	S1389
S5802 (c)	S3916
S5802 (d)	S5079
S5802 (e)	S5707
S5802 (f)	S5760
S5802 (g)	S5764
S5802 (h)	S5799
S5803	T0 S5802 T0

2.2 Use Case 2 - Microsoft (MS) PowerPoint in Windows 10

Use Case 2 is centered around different ways of starting Microsoft (MS) PowerPoint. The first scenario (A) is starting MS PowerPoint from the Microsoft Menu and creating a presentation while the next three scenarios (B - D) are modifying that same presentation but starting MS PowerPoint from the Microsoft Menu, File, Explorer, and the Command Line respectively. The next three scenarios (E - G) open a web browser (MS Edge, Google Chrome and Mozilla Firefox respectively) and then open MS PowerPoint by clicking on an appropriate presentation in the browser. Finally, the last scenario (H) opens MS PowerPoints from the Microsoft Menu, opens two presentations and cut and pastes information between them.

For Use Case 2, the term lists are identical between the DVSMT and SMTc datasets. This indicates that they are composed exclusively of terms and single object strings (i.e., strings generated by SOSLA and not by DVSMT or SMTc). As a result, there is only one set of tables for the two datasets. The one exception is Use Case 2b. Rather than creating two different tables for the Sequence of Terms and Strings, this one scenario is shown twice, once for each algorithm.

Term/String	Use Case 2 Description
T2	Modify File ^.*\windows/.*\$
T35	Modify File ^.*\$
T50	Access File ^.*\windows/.*\$
T56	System Modify Registry Key ^.*\$
T57	System Modify Registry Key ^.*\windows/.*\$
T174	User Modify Registry Key ^.*\windowsupdate/.*\$
T212	Change Metainfo File ^.*\windows/.*\$
T216	Change Metainfo File ^.*\$
T238	Change Metainfo File ^.*\users/user/appdata/local/microsoft/.*\$
T241	User Modify Registry Key ^.*\en-us/.*\$
T242	User Modify Registry Key ^.*\windows/.*\$
T243	User Modify Registry Key ^.*\windows/.*\explorer/.*\$
T329	User Access File ^.*\$
T354	Mozilla Firefox Visit URL ^.*\$
S381	Create and Access File ^.*\$
S414	Modify, Access and Change Metainfo File ^.*\$
S417	Create and Access File ^.*\Windows/.*\$
S429	Create, Modify, Access, and Change Metainfo File ^.*\users/user/appdata/local/microsoft/.*\$
S512	Modify and Access File ^.*\users/user/appdata/local/.*\$
S534	Create, Modify and Access File ^.*\\$orphanfiles/.*\$
S547	Modify, Access and Change Metainfo File ^.*\users/user/appdata/local/microsoft/.*\$

Term/String	Description
S557	Modify and Change Metainfo File ^.*\windows\.*\$
S593	Create, Modify, Access, and Change Metainfo File ^.*\users\user\appdata\local\temp\.*\$
S599	Create, Modify, Access, and Change Metainfo File ^.*\windows\.*\$
S603	Modify, Access and Change Metainfo File ^.*\windows\.*\$
S611	Modify and Change Metainfo File ^.*\users\user\appdata\local\.*\$
S613	Modify and Change Metainfo File ^.*\programdata\microsoft\.*\$
S621	Create, Modify, Access and Change Metainfo File ^.*\windows\prefetch\.*\$
S625	Modify, Access, and Change Metainfo File ^.*\programdata\microsoft\.*\$
S635	Create, Modify, Access and Change Metainfo File ^.*\$
S644	Create, Modify, Access and Change Metainfo File ^.*\users\user\appdata\local\.*\$
S646	Create, Modify, Access, and Change Metainfo File ^.*\users\user\appdata\.*\microsoft\.*\$
S647	Create. Access and Change Metainfo File ^.*\users\user\appdata\.*\microsoft\.*\$
S651	Modify and Change Metainfo File ^.*\users\user\appdata\local\microsoft\.*\$
S656	Modify and Change Metainfo File ^.*\users\user\appdata\.*\microsoft\.*\$
S657	Create, Modify, Access, and Change Metainfo File ^.*\\$orphanfiles\.*\$

Term/String	Description
S658	Modify, Access, Change Metainfo File ^.*\users/user/appdata/.*/microsoft/.*\$
S661	Modify, Access and Change Metainfo File ^.*\users/user/appdata/local/.*\$
S719	Access and Change Metainfo File ^.*\windows/.*\$
S803	Modify and Change Metainfo File ^.*\windows/prefetch/.*\$
T867	Create File ^.*\windows/program files/.*\$
T873	Access File ^.*\windows/program files/windowspowershell/modules/.*\$

Scenario	Sequence of Terms and Strings to Start Microsoft PowerPoint
UC2A	174 56 547 429 174 603 621 803 603 599 803 557 593 658 646 241 625 651 803 56 658 174 56
UC2B	212 599 417 557 56 803 803 238 243 242 613 547 429 56 547
(SMTc)	651 174 57 174 56 803 238 803 658 593 646 647 593 646 593 647 593 646 647 593 647 646 593 646 647 593 647 646 593 646 593 647 593 647 646 593 647 593 647 593 658 647 174 603 2 625 50 603 803 212 50 212 50 50 50 50 50 50 212 212 2 2 2 2 2 212 2 212 2 2 2 212 2 557 2 212 212 212 557 212 212 557 212 212 212 557 212 557 212 212 557 212 557 212 212 557 212 557 212 557 212 212 557 212 212 212 557 212 212 212 212 557 212 603 719 599 803 719 599 803 50 599 212 719 599 603 719 599 57 56 56 56 57 547 651 547 429 599 174 658 656 658 243 242 57 429 36 429 658 656 658 242 547 238 547 238 547 429 603 599 803 57

Scenario	Sequence of Terms and Strings to Start Microsoft PowerPoint
UC2B	867 417 557 56 803 803 238 243 242 613 547 429 56 547
(DVSMT)	651 174 57 174 56 803 238 803 658 593 646 647 593 646
	593 647 593 646 647 593 647 646 593 646 647 593 647
	646 593 646 593 647 593 647 646 593 647 593 647 593
	658 647 174 603 2 625 50 603 803 212 50 212 50 50 50 50 50
	50 212 212 2 2 2 2 2 2 212 2 212 2 2 2 2 212 2 557
	2 212 212 212 557 212 212 557 212 212 212 557 212
	557 212 212 557 212 557 212 212 557 212 557 212 557
	212 212 557 212 212 212 557 212 212 212 212 557 212
	603 719 599 803 719 599 803 50 873 719 599 603 719 599
	57 56 56 56 57 547 651 547 429 599 174 658 656 658 243
	242 57 429 36 429 658 656 658 242 547 238 547 238 547
	429 603 599 803 57
UC2C	625 651 613 547 429 242 803 242 174 803 238 803 242 329 803 242 658 656
	658 242 243 242 174 57 658 656 658 656 242 56 803 803 56 611 56
UC2E	803 56 613 803 803 56 429
UC2F	242 381 635 381 243 35 593 174 657 613 547 651 414 174 57 657 534
	657 656 216 242 658 646 656 658 656 658 646 243 56 599 803
UC2H	593 56 803 603 599 803 56 611 557 174 56 803

ID	Use Case 2 Production Rule for DVSMT Abstraction
S1	T547 T429 T174 T603 T621
S2	T174 T56 S1
S3	T603 T599 T803 T557 T593 T658 T646 T241 T625
S4	S2 T803 S3
S5	S4 T651 T803
S6	T803 T238 T803
S9	T174 T56
S14	T593 T646 T647 T593 T646 T593 T647 T593 T646 T647 T593 T647 T646 T593 T646 T647 T593 T647 T646 T593 T646 T593 T647 T593 T647 T646 T593 T647 T593 T647 T593 T658 T647 T174 T603 T2 T625 T50 T603 T803 T212 T50 T212 T50 T50 T50 T50 T50 T50 T212 T212 T2 T2 T2 T2 T2 T2 T212 T2 T212 T2 T2 T2 T2 T212 T2 T557 T2 T212 T212 T212 T557 T212 T212 T557 T212 T212 T212 T557 T212 T557 T212 T212 T557 T212 T557 T212 T212 T557 T212 T557 T212 T557 T212 T212 T557 T212 T212 T212 T557 T212 T212 T212 T212 T557 T212 T603 T719 T599 T803 T719 T599 T803 T50 T873 T719 T599 T603 T719 T599 T57 T56 T56 T56 T57 T547 T651 T547 T429 T599 T174 T658 T656 T658 T243 T242 T57 T429 T36 T429 T658 T656 T658 T242 T547 T238 T547 T238 T547 T429 T603 T599 T803 T57
S19	T656 T658 T242 T243 T242 T174 T57 T658 T656 T658 T656 T242 T56 T803 T803 T56 T611 T56
S24	S5 T56

ID	Use Case 2 Production Rules for DVSM T Abstraction
S26	S24 T658 S9
S38	T174 T57 T174
S68	T238 T243 T242 T613 T547 T429 T56 T547
S247	T867 T417 T557
S367	S247 T56 T803
S427	S367 T803 S68
S457	S427 T651 S38
S458	S457 T56 S6
S464	S458 T658 S14
S606	T625 T651 T613 T547 T429 T242
S668	T803 T242 T174 T803 T238 T803 T242 T329
S669	S606 S668
S699	T803 T242
S700	S669 S699
S711	S700 T658 S19
S716	T803 T56 T613 T803
S720	S716 T803
S738	S720 T56 T429
S745	T35 T593 T174 T657
S746	T0
S747	T242 T381 T635 T381 T243 S745 T613 S746 T547
S752	S747 T651 T414 T174 T57 T657 T534 T657 T656 T216 T242
S760	S752 T658 T646 T656 T658 T656 T658 T646 T243 T56 T599 T803
S768	T603 T599

ID	Use Case 2 Production Rules for DVSMT Abstraction
S769	T56 T803 S768
S770	T593 S769
S781	S770 T803 T56 T611 T557 T174 T56 T803
S782	S781
S791 (a)	S26
S791 (b)	S464
S791 (c)	S711
S791 (d)	S738
S791 (e)	S760
S791 (f)	S782
S792	T0 S791 T0

ID	Use Case 2 Production Rules for SMTC Abstraction
S1	T547 T429 T174 T603 T621
S2	T174 T56 S1
S3	T603 T599 T803 T557 T593 T658 T646 T241 T625
S4	S2 T803 S3
S5	S4 T651 T803
S6	T803 T238 T803
S9	T174 T56
S13	T593 T646 T647 T593 T646 T593 T647 T593 T646 T647 T593 T647 T646 T593 T646 T647 T593 T647 T646 T593 T646 T593 T647 T593 T647 T646 T593 T647 T593 T647 T593 T658 T647 T174 T603 T2 T625 T50 T603 T803 T212 T50 T212 T50 T50 T50 T50 T50 T50 T212 T212 T2 T2 T2 T2 T2 T2 T212 T2 T212 T2 T2 T2 T2 T212 T2 T557 T2 T212 T212 T212 T557 T212 T212 T557 T212 T212 T212 T557 T212 T557 T212 T212 T557 T212 T557 T212 T212 T557 T212 T557 T212 T557 T212 T212 T557 T212 T212 T212 T557 T212 T212 T212 T212 T557 T212 T603 T719 T599 T803 T719 T599 T803 T50 T599 T212 T719 T599 T603 T719 T599 T57 T56 T56 T56 T57 T547 T651 T547 T429 T599 T174 T658 T656 T658 T243 T242 T57 T429 T36 T429 T658 T656 T658 T242 T547 T238 T547 T238 T547 T429 T603 T599 T803 T57
S17	T656 T658 T242 T243 T242 T174 T57 T658 T656 T658 T656 T242 T56 T803 T803 T56 T611 T56
S21	S5 T56
S23	S21 T658 S9

ID	Use Case 2 Production Rules for SMTC Abstraction
S32	T174 T57 T174
S56	T238 T243 T242 T613 T547 T429 T56 T547
S199	T212 T599 T417 T557
S295	S199 T56 T803
S343	S295 T803 S56
S367	S343 T651 S32
S368	S367 T56 S6
S373	S368 T658 S13
S486	T625 T651 T613 T547 T429 T242
S536	T803 T242 T174 T803 T238 T803 T242 T329
S537	S486 S536
S561	T803 T242
S562	S537 S561
S571	S562 T658 S17
S575	T803 T56 T613 T803
S579	S575 T803
S594	S579 T56 T429
S602	T242 T381 T635 T381 T243 T35 T593 T174 T657 T613 T547 T651 T414 T174 T57 T657 T534 T657 T656 T216 T242
S610	T658 T646 T656 T658 T656
S611	S602 S610
S616	T56 T599 T803
S617	S611 T658 T646 T243 S616
S618	S617

ID	Use Case 2 Production Rules for SMTC Abstraction
S625	T603 T599
S626	T56 T803 S625
S627	T593 S626
S637	S627 T803 T56 T611 T557 T174 T56 T803
S638	S637
S646 (a)	S23
S646 (b)	S373
S646 (c)	S571
S646 (d)	S594
S646 (e)	S618
S646 (f)	S638
S647	T0 S646 T0

2.3 Use Case 3 - Microsoft (MS) Excel in Windows 10

Use Case 3 is centered around different ways of starting Microsoft (MS) Excel. The first scenario (A) is starting MS Excel from the Microsoft Menu and creating a spreadsheet while the next three scenarios (B - D) are modifying that same spreadsheet but starting MS Excel from the Microsoft Menu, File, Explorer, and the Command Line respectively. The next three scenarios (E - G) open a web browser (MS Edge, Google Chrome and Mozilla Firefox respectively) and then open MS Excel by clicking on an appropriate spreadsheet in the browser. Finally, the last scenario (H) opens MS Excel from the Microsoft Menu, opens two spreadsheets and cut and pastes information between them.

Term/String	Use Case 3 Description
T2	Modify File ^.*\windows\.*\$
T58	Modify Registry Key ^.*\Microsoft\.*\$
T184	
T241	Change Metainfo File ^.*\windows\.*\$
T256	Change Metainfo File ^.*\ProgramData\Microsoft\Windows\.*\$
T289	Access File ^.*\Users\user\AppData\Local\Microsoft\OneDrive\.*\$
T297	Modify File ^.*\Program Files (x86)\.*\$
T325	Change Metainfo File ^.*\Windows\.*\en-US\.*\$
S403	Create, Modify, Access, Change Metainfo, Create and Modify File ^.*\Windows\.*\$
S458	Modify, Access, Change Metainfo File ^.*\Users\user\AppData\Local\Temp\.*\$
S543	Modify, Access, Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S553	Modify, Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
S556	Modify and Change Metainfo File ^.*\ProgramData\Microsoft\.*\$
S658	Modify and Change Metainfo File ^.*\Windows\.*\$
S662	Modify, Access and Change Metainfo File ^.*\Windows\.*\$
S688	Modify, Access and Change Metainfo File ^.*\ProgramData\Microsoft\.*\$
S750	Access and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\Windows\.*\$

Term/String	Use Case 3 Description
S753	Modify and Change Metainfo File ^.*\Users/user/Appdata/Local/Microsoft/Windows/.*\$
S756	Modify and Change Metainfo File ^.*\Users/user/Appdata/.*\Microsoft/Windows/.*\$
S759	Modify and Change Metainfo File ^.*\Users/user/Appdata/.*\Microsoft/.*\$
S762	Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/Microsoft/Windows/.*\$
S767	Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S803	Access and Change Metainfo File ^.*\Windows/.*\$
S847	Modify, Access and Change Metainfo File ^.*\Program Files/Common Files/.*\$
S854	Modify and Change Metainfo File ^.*\Appdata/Local/.*\$
S898	Modify and Change Metainfo File ^.*\Windows/Prefetch/.*\$
T907	Create File ^.*\$
T908	Access File ^.*\$
T909	Modify File ^.*\$
T910	Modify Registry Key ^.*\$
T912	Change File ^.*\$
T916	
S923	Modify, Access, Change Metainfo, Modify, Access, and Change MetaInfo File ^.*\Users/user/Appdata/Local/Microsoft/OneDrive/.*\$
S936	Access and Change Metainfo File ^.*\Users/user/Appdata/.*\$
S948	Create and Access File ^.*\$

Term/String	Use Case 3 Description
S957	Modify, Change Metainfo, Modify, Change Metainfo File ^.*\$
S983	Create, Modify, Access, Create, Modify, Access File ^.*\Windows\.*\$
S986	Create and Create File ^.*\Windows\.*\$
S1031	Create and Create File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S1042	Create, Modify, Access, Change Metainfo, Create, and Modify File ^.*\Users\user\AppData\.*\$
S1097	Create and Create File ^.*\Users\user\AppData\.*\Microsoft\Windows\.*\$
S1108	Access and Access File ^.*\Windows\.*\$
S1124	Access, Modify, Access, and Change Metainfo File ^.*\Windows\.*\$
S1140	Create and Create File ^.*\$
S1159	Access and Access File ^.*\$
S1191	Create, Modify, Access, Change Metainfo, Create, and Modify File ^.*\Users\user\AppData\.*\Microsoft\Windows\.*\$
S1192	Modify and Change Metainfo File ^.*\$
S1199	Create, Modify, Access, Change Metainfo, Create, and Access File ^.*\Windows\.*\$
S1217	Modify, Access, Change Metainfo, Modify, Access, and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\Windows\.*\$
S1227	Modify and Change Metainfo File ^.*\Windows\.*\$
S1274	Create, Modify, Access, Change Metainfo, Create, and Modify File ^.*\Users\user\AppData\Local\.*\$
S1285	Create, Modify, Access, Change Metainfo, Create, and Modify File ^.*\Users\user\AppData\Local\Temp\.*\$

Term/String	Use Case 3 Description
S1288	Access and Change Metainfo File ^.*\Users/user/Appdata/Local/Temp/.*\$
S1310	Create, Access, Create, Modify, and Access File ^.*\$
S1444	Access, Modify, Access, and Change File ^.*\Users/user/Appdata/.*/Microsoft/Windows/.*\$
S1451	Create, Access, Change Metainfo, Create, Access, and Change Metainfo File ^.*\Windows/.*\$
S1455	Create, Access, Create and Access File ^.*\$
S1529	Modify, Access, Change Metainfo, Modify, Access, and Change Metainfo File ^.*\Users/user/Appdata/.*/Microsoft/.*\$
S1530	Create, Modify, Access, Change Metainfo, Create, and Access File ^.*\Users/user/Appdata/Local/Microsoft/.*\$
S1560	Modify, Change, Modify and Change File ^.*\Windows/.*\$
S1589	Modify, Access, Change Metainfo, Modify, and Access File ^.*\Users/user/Appdata/Local/.*\$
S4420	S1285 + S1288
S5212	S1042 + S936
S7206	S403 + S803
S9341	S1191 + S750

Scenario	Sequence of Terms and Strings to Start Microsoft Excel (DVSMT Abstraction)
UC3A	58 898 898 910 241 325 241 241 241 241 256 285 241 325 241 285 241 241 325 241 241 297 241 403 803 241 325 241 241 910 241 741 241 910 325 241 325 241 662 403 803 898 403 803 403 803 403 803 403 803 403 803 403 803 403 803 403 803 403 803 403 803 403 803 658 910 403 803 241 910 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 1560 241 1227 58 241 1108 658 1108 58 241 58 51 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58
UC3B	184 910 184 184 898 898 556 58 898 762 58 688 756 184 847 1444 756 184 983 241 910 662 241 898 923

Scenario	Sequence of Terms and Strings to Start Microsoft Excel (DVSMT Abstraction)
UC3C	184 916 184 184 58 184 185 58 184 58 185 910 910 184 58 58 910 58 910
UC3D	898 898 184 58 756 184 756 543 1529 759 847 1444 854 756 898 184 898 662 403 803 910 898 658
UC3E	1455 948 1310 948 1192 184 912 909 1192 1285 1288 184 898 910 898 58 553 898 1031 912 184 1217 1097 1191 750 753 58 662 403 803 898 1274 767 1589 658
UC3F	58 910 553 184 1140 907 184 1285 1288 184 58 898 1031 847 756 1444 1191 750 184 543 1529 759 756 898 184 662 403 803 898 58 658 910 662 1124 241 58 184 898 910 1159 908 458 1285 1288 957 1192
UC3G	898 58 184 898 1199 58 1530 898 910 1199 662 403 803 898 658 1558 910 688 58 1042 936
UC3H	184 688 184 58 898 898 898 184 986 1108 898 662 1560 658 898 1451 1451

Scenario	Sequence of Terms and Strings to Start Microsoft Excel (SMTC Abstraction)
UC3A	58 898 898 910 241 325 241 241 241 241 256 285 241 325 241 285 241 241 325 241 241 297 241 7206 241 325 241 241 910 241 741 241 910 325 241 325 241 662 7206 898 7206 7206 658 910 7206 241 910 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 1560 241 1227 58 241 1108 658 1108 58 241 58 51 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 1108 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 58 241 2 58 241 58 241 58 241 58
UC3B	184 910 184 184 898 898 556 58 898 762 58 688 756 184 847 1444 756 184 983 241 910 662 241 898 923

Scenario	Sequence of Terms and Strings to Start Microsoft Excel (SMTC Abstraction)
UC3C	898 898 184 58 756 184 756 543 1529 759 847 1444 854 756 898 184 898 662 7206 910 898 658
UC3D	1455 948 1310 948 1192 184 912 909 1192 4420 184 898 910 898 58 553 898 1031 912 184 1217 1097 9341 753 58 662 7206 898 5041 1589 658
UC3E	58 910 553 184 1140 907 184 4420 184 58 898 1031 847 756 1444 9341 184 543 1529 759 756 898 184 662 7206 898 58 658 910 662 1124 241 58 184 898 910 1159 908 458 4420 957 1192
UC3F	898 58 184 898 1199 58 1530 898 910 1199 662 7206 898 658 1558 910 688 58 5212
UC3G	184 688 184 58 898 898 898 184 986 1108 898 662 1560 658 898 1451 1451
UC3H	184 916 184 184 58 184 185 58 184 58 185 910 910 184 58 58 910 58 910

ID	Use Case 3 Production Rules for DVSMT Abstraction
S1	T58 T898 T898
S2	T241 T325 T241 T241 T241 T241 T256 T285 T241 T325 T241 T285 T241 T241 T325
S3	S1 T910 S2
S5	T241 T297
S6	S3 T241 S5
S7	T403 T803 T241 T325 T241 T241 T910 T241 T741 T241 T910 T325 T241 T325 T241 T662 T403 T803

ID Use Case 3 Production Rules for DVSM T Abstraction

S8 T403 T803 T403 T803 T403 T803 T403 T803 T403 T803 T403 T803
T403 T803 T403 T803 T403 T803 T403 T803 T403 T803 T658 T910
T403 T803 T241 T910 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T1108 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T1108 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T1108 T58 T241 T58 T241 T1560 T241
T1227 T58 T241 T1108 T658 T1108 T58 T241 T58 T51 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T1108 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T2 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T2 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T2 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T2 T58 T241 T58
T241 T58 T241 T58

ID	Use Case 3 Production Rules for DVSM T Abstraction
S9	T58 T910 T553 T184 T1140 T907 T184 T1285 T1288 T184 T58 T898 T1031 T847 T756 T1444 T1191 T750 T184 T543 T1529 T759 T756 T898 T184 T662
S12	T803 T898 T58 T658
S13	T662 T1124
S15	S6 T241 S7 T898 S8
S46	T910 T662
S76	T184 T184 T898 T898 T556 T58 T898 T762 T58 T688 T756 T184 T847 T1444 T756 T184 T983
S138	T184 T688 T184 T58 T898 T898 T898 T184 T986 T1108 T898
S142	T662 T1560 T658
S202	T184 T910 S76
S232	S202 T241 S46
S254	S232 T241 T898 T923
S261	T184 T916
S262	T58 T184 T185
S263	T184
S264	T185 T910 T910
S325	T184 T184 S262 T58 S263 T58 S264 T184 T58 T58 T910 T58
S326	S261 S325
S356	S326 T910
S366	T898 T1199
S369	T184 S366 T58 T1530 T898 T910 T1199 T662 T403 T803
S371	T58 S369

ID	Use Case 3 Production Rules for DVSM T Abstraction
S372	T898 S371 T898 T658 T1558
S375	S372 T910 T688
S379	T184 T58 T756 T184 T756 T543 T1529 T759 T847 T1444 T854 T756
S380	T898 T898 S379 T898 T184 T898 T662
S383	T0
S385	T403 T803 S383 T910
S400	S380 S385 T898 T658
S401	S400
S402	S401
S410	T912 T909 T1192 T1285 T1288 T184
S411	T910 T898
S412	T553 T898 T1031 T912
S416	T1455 T948 T1310 T948 T1192 T184 S410 T898 S411 T58 S412 T184 T1217 T1097 T1191 T750 T753 T58 T662
S419	T403 T803
S430	S416 S419 T898 T1274 T767 T1589 T658
S431	S430
S432	S431
S440	T403 S12 T910 S13 T241 T58 T184 T898 T910 T1159 T908 T458 T1285 T1288 T957 T1192
S441	S9 S440
S449	T58 T1042 T936
S450	S375 S449
S451	S450
S459	S138 S142 T898 T1451 T1451

ID	Use Case 3 Production Rules for DVSMT Abstraction
S460	S459
S468 (a)	S15
S468 (b)	S254
S468 (c)	S356
S468 (d)	S402
S468 (e)	S432
S468 (f)	S441
S468 (g)	S451
S468 (h)	S460
S469	T0 S468 T0

ID	Use Case 3 Production Rules for SMTC Abstraction
S1	T58 T898 T898
S3	T241 T325 T241 T241 T241 T241 T256 T285 T241 T325 T241 T285 T241 T241 T325
S4	S1 T910 S3
S6	T241 T297
S7	S4 T241 S6
S8	T7206 T241 T325 T241 T241 T910 T241 T741 T241 T910 T325 T241 T325 T241 T662 T7206
S9	T58 T910 T553 T184 T1140 T907 T184 T4420 T184 T58 T898 T1031 T847 T756 T1444 T9341 T184 T543 T1529 T759 T756 T898 T184 T662
S12	T898 T58 T658
S13	T662 T1124
S14	T58 T184 T898
S16	S7 T241 S8

ID Use Case 3 Production Rules for SMTC Abstraction

S20 T7206 T7206 T658 T910 T7206 T241 T910 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241
T58 T241 T1108 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108
T58 T241 T58 T241 T1560 T241 T1227 T58 T241 T1108 T658 T1108
T58 T241 T58 T51 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T1108 T58 T241 T58 T241 T58 T241
T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T1108 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T2 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T2 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T58 T241 T58 T241 T58 T241 T58
T241 T58 T241 T58 T241 T2 T58 T241 T58 T241 T58 T241 T58

ID	Use Case 3 Production Rules for SMTC Abstraction
S21	S16 T898 S20
S30	T5041 T1589 T658
S51	T910 T662
S83	T184 T184 T898 T898 T556 T58 T898 T762 T58 T688 T756 T184 T847 T1444 T756 T184 T983
S147	T184 T916 T184 T184 T58 T184 T185 T58 T184 T58 T185
S151	S147 T910 T910 T184 T58 T58 T910 T58
S155	T184 T688 T184 T58 T898 T898 T898 T184 T986 T1108 T898
S166	T662 T1560 T658
S167	S155 S166
S233	T184 T910 S83
S265	S233 T241 S51
S281	S265 T241
S287	S281 T898 T923
S296	T58 T184
S297	T898 S296 T898 T1199 T58 T1530 T898
S301	S297 T910 T1199 T662
S303	T898 T658 T1558
S309	T7206 S303 T910 T688
S310	S301 S309
S444	T898 T898
S446	T58 T756
S447	T756 T543 T1529 T759 T847 T1444 T854

ID	Use Case 3 Production Rules for SMTC Abstraction
S448	T898
S513	T184 S446 T184 S447 T756 S448 T184 T898 T662 T7206
S514	S444 S513
S546	S514 T910
S556	S546 T898 T658
S562	T912 T909 T1192 T4420 T184
S563	T910 T898
S564	T553 T898 T1031 T912
S568	T1455 T948 T1310 T948 T1192 T184 S562 T898 S563 T58 S564 T184 T1217 T1097 T9341 T753 T58 T662
S587	S568 T7206
S600	S587 T898 S30
S603	T7206 S12 T910 S13 T241 S14 T910 T1159 T908 T458 T4420 T957 T1192
S604	S9 S603
S611	T58 T5212
S612	S310 S611
S620	S167 T898 T1451 T1451
S624	S151 T910
S626	S624

ID	Use Case 3 Production Rules for SMTC Abstraction
S636	S626
S640 (a)	S21
S640 (b)	S287
S640 (c)	S556
S640 (d)	S600
S640 (e)	S604
S640 (f)	S612
S640 (g)	S620
S640 (h)	S636
S641	T0 S640 T0

2.4 Use Case 4 - Google Chrome in Windows 10

Use Case 4 is centered around different ways of using Web Browsing. The first scenario (A) simply visits a website. The second and third scenario (B - C) visit a website and download content. Finally, the remaining scenarios (D - F) perform different applications including Facebook Instant Messaging, Twitter and Instagram.

Term/String	Use Case 4 Description
T58	Modify Registry Key ^.*\$
T60	Modify Registry Key ^.*\Windows\.*\$
T175	
T219	Change Metainfo File ^.*\Windows\.*\$
T248	Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
T252	Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
T253	
T261	Change Metainfo File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
T268	Change Metainfo File ^.*\Windows\.*\WindowsUpdate\.*\$
S401	Access and Change Metainfo File ^.*\Users\user\AppData\Local\.*\$
S417	Create, Modify, Access, Change Metainfo File, Create, and Access ^.*\Users\user\AppData\Local\.*\$
S431	Modify, Access and Change Metainfo File ^.*\Users\user\AppData\.*\Microsoft\.*\$
S449	Create, Access, Change Metainfo, Create, and Access File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S467	Create ^.*\Users\user\AppData\Local\.*\$
S628	Access File ^.*\Windows\.*\$
S647	Modify and Change Metainfo File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S692	Access and Change Metainfo File ^.*\Windows\Prefetch\.*\$

Term/String	Use Case 4 Description
S705	Create, Modify, Access, Change Metainfo, and Create File ^.*\Users/user/Appdata/Local/.*\$
S778	Modify, Change Metainfo and Modify and Change Metainfo File ^.*\Windows/.*\$
S797	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\ProgramData/Microsoft/Windows/AppRepository/.*\$
S801	Modify and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S815	Create, Access, Create, Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S826	Modify and Change Metainfo File ^.*\Windows/.*\$
S834	Modify, Access and Change Metainfo File ^.*\Windows/.*\$
S836	Create, Modify, Access, Change Metainfo, Create and Modify File ^.*\Windows/.*\$
S837	Access and Change Metainfo File ^.*\Windows/.*\$
S854	Modify and Change Metainfo File ^.*\ProgramData/Microsoft/.*\$
S858	Create, Access, Change Metainfo, Create, Access and Change MetaInfo File ^.*\Windows/WinSxS/.*\$
S859	Modify, Access, Change Metainfo, Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/.*\Microsoft/.*\$
S871	Modify, Access and Change Metainfo File ^.*\ProgramData/Microsoft/.*\$
S892	Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$

Term/String	Use Case 4 Description
S914	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S915	Modify, Access, Change Metainfo, Modify, Access and Change Metainfo File ^.*\Windows/.*\$
S921	Create, Modify, Access, Create, Modify, and Access File ^.*\Users/user/Appdata/Local/.*\$
S924	Create, Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S931	Access and Change Metainfo File ^.*\Users/user/Appdata/.*\Microsoft/.*\$
S935	Create, Modify, Access, Change Metainfo, Create and Modify File ^.*\Users/user/Appdata/Local/.*\$
S936	Create, Modify, Access, Change Metainfo, Create and Modify File ^.*\Users/user/Appdata/Local/Microsoft/.*\$
S952	Create, Modify, Access, Change Metainfo, Create and Modify File ^.*\Users/user/Appdata/Local/Temp/.*\$
S954	Access and Change Metainfo File ^.*\Users/user/Appdata/Local/Temp.*\$
S969	Access and Access File ^.*\Users/user/Appdata/Local/.*\$
S1014	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\Windows/.*\$
S1101	Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S1105	Access, Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$

Term/String	Use Case 4 Description
S1223	Modify, Change Metainfo, Modify and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S1262	Modify, Access, Modify, and Access File ^.*\Users/user/Appdata/Local/.*\$
S1263	Modify, Access, Change Metainfo, Modify, Access and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S1272	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\ProgramData/Microsoft/Windows/AppRepository/Packages/.*\$
S1320	Modify, Access and Change Metainfo File ^.*\OrphanFiles/.*\$
S1335	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\Program Files/WindowsApps/.*\$
S1380	Access, Create and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S1381	Modify, Access, Change Metainfo, Modify, Access and Change Metainfo File ^.*\ProgramData/Microsoft/.*\$
S1394	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\Windows/.*\WindowsUpdate/.*\$

Term/String	Use Case 4 Description
S1395	Modify, Access, Change Metainfo, Modify and Access File ^.*\Users/user/Appdata/Local/.*\$
S1413	Create, Access, Change Metainfo, Create, Access and Change Metainfo File ^.*\$
S1418	Create, Access, Change Metainfo, Create and Access File ^.*\Users/user/Appdata/Local/.*\$
S1429	Access, Modify, Change Metainfo, Modify and Change Metainfo File ^.*\Users/user/Appdata/Local/.*\$
S1430	Access and Modify File ^.*\Users/user/Appdata/.*\Microsoft/.*\$
S1431	Change Metainfo and Change Metainfo File ^.*\Users/user/Appdata/.*\Microsoft/.*\$
T1531	
T1601	Access File ^.*\$
T1612	Modify Registry Key ^.*\$
T1640	Access File ^.*\Windows/diagnostics/system/.*\$
T1648	Access File ^.*\Windows/Boot/.*\$
T1649	
T1650	Access File ^.*\Windows/Help/Windows/.*\$

Scenario	Sequence of Terms and String for Google Chrome (DVSMT Abstraction)
UC4B	1531 175 1531 692 692 692 801 692 1101 1105 892 952 954 801 1262 248 692 175 1105 248 1101 1105 892 1395 1101 1105 892 1101 1105 892 1101 1105 892 921 248 1612 969 801 1105 892 403 719 1185 253 1101 1105 892 815 1531 801 1395 1101 1612 1531 1395 1263 175 801 1531 1395 801 1531 1395 692 951 834 219 915 826 60 58 1263 1101 1263 248 1263 801 1101 1263 248 1531 801 248 1395 815 801 935 401 248 815 801 801 1531 801 1101 1612 801 1263 1395 268 1263 705 1101 705 1101 1395 795 417 815 1531 801
UC4C	801 1101 1105 892 1014 692 801 1263 1335 1014 1014 1014 1014 1014 1380 1014 1014 1380 1014 1380 1014 175 1105 248 1101 1105 892 801 1014 1014 1014 871 1381 1223 1014 1014 692 1014 801 1014 1394 871 1381 1014 1101 815 854 1101 1105 892 1223 801 801 1223 1014 1223 801 692 1014 914 1014 1014 1320 871 1381 871 1381 692 1014 1014 1101 1105 892 1014 1014 1101 1105 892 1101 1105 892 1612 1531 248 1014 1014 431 859 253 1014 969 1014 628 1014 1223 778 175 1531 1014 914 1531 1101 1105 892 1014 1014 1649 1101 801 1531 1014 660 1014 219 1014 1101 1263 1014 801 1014 1014 1014 1014 1014 1014 1014 1014 1014 801 1014 1014 1014 1413 1394 1014 1394 1014 1335 826

Scenario	Sequence of Terms and String for Google Chrome (DVSMT Abstraction)
UC4D	58 692 836 837 253 175 801 1101 1105 892 1101 801 1105 248 175 1101 1105 892 801 1101 1105 892 248 692 1263 248 692 1101 1105 892 1101 1105 892 1101 1105 892 1101 1105 892 1531 801 921 248 1612 936 931 252 1601 248 1263 175 248 801 692 826 647
UC4E	797 797 797 797 797 797 797 797 914 797 253 175 797 914 801 1101 1105 892 175 1105 248 1101 1105 892 692 449 797 692 797 261 797 1101 1105 892 1101 1105 892 797 1612 1101 1105 892 1101 1105 892 1531 1101 1105 892 801 921 248 1612 936 931 692 858 692 253 1263 175 924 815 1531 924 467 1531 815 1101 1531 1640 914 914 801 1380 1223 801 871 1223 801 1223 1223 1223 801 914 801 692 1531 1101 1263 1650 1648 1640 1531 1649 801 1531 1263 801 797 1272 914 1531 1272 1531 1101 858 1612 1531 914 1612 1531 1101 1531
UC4F	914 914 914 692 1418 914 248 914 253 914 175 692 914 1014 692 914 801 1101 1105 892 692 1262 892 175 1105 248 801 248 1101 1105 248 914 914 1014 1014 1101 1105 892 1335 914 1531 815 1531 815 801 1101 1105 892 1612 1531 248 1101 1105 892 936 931 815 1014 1101 1105 892 801 1429 175 1101 1105 892 1101 1531 801 1430 431 1431 1101 815 1531 253 801 692 1101 1263 1380 1223 1014 1014 1014 1014 1014 826 801 1223 801 1223 801 1014 1014 1014 1014 1014 1014 1014 1014 1014 1014 1014 1014 1014 1014 801 1014 1014

Scenario	Sequence of Terms and String for Google Chrome (SMTC Abstraction)
UC4B	2987 175 2987 692 692 692 801 692 1101 1105 892 3113 801 1262 248 692 1105 248 1101 1105 892 1395 1101 1105 892 1101 1105 892 1101 1105 892 921 248 2987 969 801 1105 892 403 719 1185 253 1101 1105 892 815 2987 801 1395 1101 2987 1395 1263 175 801 2987 1395 801 2987 1395 692 834 219 915 826 60 58 1263 1101 1263 248 1263 801 1101 1263 248 2987 801 248 1395 815 801 935 401 248 815 801 801 2987 801 1101 2987 801 1263 1395 1263 705 1101 705 1101 1395 417 815 2987 801
UC4C	801 1101 1105 892 692 801 1263 1335 1380 1380 1380 1105 248 1101 1105 892 801 871 1381 1223 692 801 871 1381 1101 815 854 1101 1105 892 1223 801 801 1223 1223 801 692 914 871 1381 871 1381 692 1101 1105 892 1101 1105 892 1101 1105 892 2987 248 431 859 253 969 1223 175 2987 914 2987 1101 1105 892 2987 815 1101 801 2987 1101 1263 801 801 1413 1335 826

Scenario	Sequence of Terms and String for Google Chrome (SMTc Abstraction)
UC4D	58 692 3091 253 175 801 1101 1105 892 1101 801 1105 248 1101 1105 892 801 1101 1105 892 248 692 1263 248 692 1101 1105 892 1101 1105 892 1101 1105 892 1101 1105 892 2987 801 921 248 2987 3243 252 248 1263 175 248 801 692 826 647
UC4E	797 797 797 797 797 797 797 797 914 797 253 175 797 914 801 1101 1105 892 1105 248 1101 1105 892 692 449 797 692 797 261 797 1101 1105 892 1101 1105 892 797 2987 1101 1105 892 1101 1105 892 2987 1101 1105 892 801 921 248 2987 3243 692 692 253 1263 175 924 815 2987 924 467 2987 815 1101 2987 801 2987 914 914 801 1380 1223 801 871 1223 801 1223 1223 1223 801 914 801 692 2987 1101 1263 2987 801 815 2987 801 2987 2987 3379 2987 1263 801 797 914 5593 2987 1101 2987 914 2987 1101 2987
UC4F	914 914 914 692 1418 914 248 914 253 914 175 692 914 692 914 801 1101 1105 892 692 1262 892 1105 248 801 248 1101 1105 248 914 914 1101 1105 892 1335 914 2987 815 2987 815 801 1101 1105 892 2987 248 1101 1105 892 3243 815 1101 1105 892 801 1429 175 1101 1105 892 1101 2987 801 1430 431 1431 1101 815 2987 253 801 692 1101 1263 1380 1223 826 801 1223 801 1223 801 801

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S1	T1531 T175 T1531 T692 T692
S3	S1 T692
S5	T692 T1101 T1105 T892 T952 T954 T801 T1262 T248 T692
S8	S3 T801 S5
S10	T1105 T248 T1101
S11	T892 T1395
S13	S10 T1105 S11
S14	S8 T175 S13
S17	T892 T801
S25	T1101 T1105 S17
S29	T1105 T248
S33	T797 T914 T801
S37	T1263 T1335 T1014 T1014 T1014 T1014 T1014 T1380 T1014 T1014 T1380 T1014 T1380 T1014
S52	T1101
S53	T1105
S69	T1101 T1105 T892 S52 T801 S53 T248
S84	T801 T1101 T1105 T892 T1014
S86	S84 T692
S100	S86 T801 S37
S109	S100 T175 S29
S188	T58 T692 T836 T837 T253 T175
S228	S188 T801 S69
S236	S228 T175 S25

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S292	T797 T797 T797 T797 T797 T797 T797 T797 T797 T914 T797 T253
S307	S292 T175 S33
S312	T914 T914 T914 T692 T1418 T914 T248 T914 T253 T914 T175
S316	T914 T1014
S333	T692 S316 T692 T914 T801 T1101
S336	S312 S333
S339	T1105 T892 T692 T1262
S343	T892 T175
S345	S339 S343
S347	S336 S345
S348 (a)	S14 T1101
S348 (b)	S109 T1101
S348 (c)	S235 T1101
S348 (d)	S307 T1101
S348 (e)	s347
S349	T1101 T1105 T892 T1101 T1105 T892 T921 T248 T1612 T969
S351	T1105 T892 T403 T719 T1185 T253
S352	S349 T801 S351
S356	T892 S352
S357	T1014 T1394 T871 T1381 T1014
S363	T801 T1014 T1014 T1014 T871 T1381 T1223 T1014 T1014 T692 T1014
S370	S363 T801 S357
S374	T892 S370

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S387	T248 T692 T1263 T248 T692 T1101 T1105 T892 T1101 T1105 T892 T1101
S394	T1105 T892
S395	S387 S394
S399	T892 S395
S400	T0
S403	T175 T1105 T248 S400 T1101 T1105 T892 T692 T449 T797 T692 T797 T261 T797 T1101
S411	S403 S394
S416	T892 S411
S417	T1101 T1105
S418	T914 T914 T1014 T1014
S421	T248 T801 T248 S417 T248 S418 T1101 T1105 T892 T1335 T914 T1531 T815 T1531 T815
S429	S421 T801
S435 (a)	S356
S435 (b)	S374
S435 (c)	S399
S435 (d)	S416
S435 (e)	S429
S436	T1531 T801 T1395
S437	S394 T815 S436
S438	T1612 T1531 T1395 T1263 T175
S439	T1105 T892 T797 T1612 T1101 T1105 T892
S443	S437 T1101 S438
S445	T1531 T1395 T801 T1531 T1395

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S446	T1531 T1101 T1105 T892
S447	S439 T1101 T1105 T892 S446
S448	S447
S450	S443 T801 S445
S454	T951 T834 T219 T915 T826 T60 T58 T1263
S456	T1263 T248 T1263 T801
S457	T1263 T248
S458	T1395 T815 T801 T935 T401 T248 T815 T801 T801
S459	T801 T1101 T1612 T801 T1263 T1395 T268 T1263 T705
S460	T1395 T795 T417 T815
S461	S454 T1101 S456 T1101 S457 T1531 T801 T248 S458 T1531 S459 T1101 T705 T1101 S460 T1531 T801
S462	T1014 T660 T1014 T219 T1014 T1101 T1263 T1014 T801 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T801 T1014 T1014 T1014 T1413 T1394 T1014 T1394 T1014 T1335 T826
S466	T1105 T892 T1014 T1014 T1649
S471	T1014 T914 T1531
S480	T1014 T1014 T431 T859 T253 T1014 T969 T1014 T628 T1014 T1223 T778 T175
S529	T1105 T892 T1101 T1105 T892 T1612
S594	T1105 T892 T1014 T1014
S723	T1014 T914 T1014 T1014 T1320 T871 T1381 T871 T1381 T692 T1014 T1014

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S980	S723 T1101 S594 T1101 S529 T1531 T248 S480 T1531 S471 T1101 S466 T1101 T801 T1531 S462
S1239	T826 T647
S1242	T858 T692 T253 T1263 T175 T924 T815 T1531 T924 T467 T1531 T815 T1101 T1531 T1640 T914 T914 T801 T1380 T1223 T801 T871 T1223 T1223 T1223 T1223 T801 T914 T801 T692 T1531 T1101 T1263 T1650 T1648 T1640 T1531 T1649 T801 T1531 T1263 T801 T797 T1272 T914 T1531 T1272 T1531 T1101 T858 T1612 T1531 T914 T1612 T1531 T1101 T1531
S1247	T1105 T892 T1223 T801 T801 T1223 T1014 T1223
S1285	T815 T854
S1293	S1285 T1101 S1247
S1297	S1293 T801
S1300	T1105 T892 T1612
S1302	T1531 T248
S1303	S1300 S1302
S1306	S1303 T1101 T1105 T892 T936 T931 T815 T1014 T1101 T1105 T892
S1325	T1531 T801 T921 T248
S1326	S394 S1325
S1327	T936 T931 T252 T1601 T248
S1336	T1612 S1327 T1263 T175 T248

ID	Use Case 4 Production Rules (DVSMT Abstraction)
S1337	S1326 S1336
S1341	S1337 T801
S1344	S448 T801 T921 T248 T1612 T936 T931
S1348	T801 T1429 T175 T1101 T1105 T892 T1101 T1531 T801 T1430 T431 T1431 T1101 T815 T1531 T253 T801 T692 T1101 T1263 T1380 T1223 T1014 T1014 T1014 T1014 T1014 T826 T801 T1223 T801 T1223 T801 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T1014 T801 T1014 T1014
S1349	S1306 S1348
S1353 (a)	S450 T692 S461
S1353 (b)	S1297 T692 S980
S1353 (c)	S1341 T692 S1239
S1353 (d)	S1344 T692 S1242
S1353 (e)	S1349
S1354	T0 S348 T1105 S435 T1101 S1353 T0

ID	Use Case 4 Production Rules (SMTC Abstraction)
S1	T2987 T175 T2987 T692
S3	S1 T692 T692
S5	T692 T1101
S7 (a)	T3113
S7 (b)	T1101
S8	T892 S7 T801 T1262
S9	S8 T248 T692
S10	T248 T1101 T1105
S11	T1395 T1101 T1105 T892
S13	S3 T801 S5 T1105 S9 T1105 S10 T892 S11
S16	T1105 T892
S17	S16 T1101
S18	T815 T854
S19	S18 T1101
S21 (a)	T871 T1381 T1223
S21 (b)	T1101 T1105 T892 T248
S22	T801 S21 T692 T801 T871 T1381
S27	T801 S21 T692 T1263 T248 T692
S79	T248 T1101
S98	T892 S7 T801
S118	T1263 T1335 T1380 T1380 T1380
S572	T801 T1101 T1105 T892
S846	S572 T692

ID	Use Case 4 Production Rules (SMTC Abstraction)
S1004	S846 T801 S118 T1105 S79 T1105 T892 S22
S1664	T58 T692 T3091 T253 T175
S1926	S1664 T801 T1101 T1105 S98 T1105 S10 T892 S27
S2212	T797 T797 T797 T797 T797 T797 T797 T797 T914 T797 T253 T175 T797 T914
S2221	S2212 T801
S2228	T914 T914 T914 T692 T1418 T914 T248 T914 T253 T914 T175
S2232	T692 T914
S2234	T692 T914 T801 T1101 T1105 T892 T692 T1262 T892
S2242	T1105 T248 T801 T248
S2244	S2228 S2232 S2234 S2242
S2245	S2244
S2246	S2245
S2250 (a)	S13 T1101 S17
S2250 (b)	S1004 T1101 S19
S2250 (c)	S1926 T1101 S17
S2250 (d)	S2221 T1101 S16
S2250 (a)	S2246 T1101
S2251	T921 T248 T2987 T969
S2252	T1105 T892 T403 T719 T1185 T253
S2253	S2251 T801 S2252
S2254	T815 T2987 T801 T1395
S2255	S2253 T1101 T1105 T892 S2254
S2256	T1395 T1263
S2258	T801 T2987 T1395 T801

ID	Use Case 4 Production Rules (SMTC Abstraction)
S2260	T1395 T692 T834 T219 T915 T826 T60 T58 T1263
S2262	T1263 T248
S2263	T1101 T1263 T248 T2987
S2264	T248 T1395 T815 T801 T935 T401 T248 T815 T801 T801 T2987 T801 T1101 T2987 T801 T1263 T1395 T1263 T705 T1101 T705 T1101 T1395 T417 T815 T2987 T801
S2268	T1413 T1335 T826
S2278	T801 T2987 T1101
S2291	T914 T2987 T1101 T1105 T892 T2987 T815
S2368	T248 T431 T859 T253 T969 T1223
S2472	S2255 T1101 T2987 S2256
S2473	S2472 T175 S2258
S2474	S2473 T2987 S2260
S2475	S2474 T1101 S2262 T1263 T801 S2263 T801 S2264
S2476	T892 S2475
S2678	T1101 T1105 T892
S3092	T692 T914 T871 T1381 T871 T1381 T692
S3920	T1223 T801 T801 T1223 T1223
S4748	S3920 T801 S3092
S5162	S4748 T1101 T1105 T892 S2678
S5266	S5162 T1101 S16 T2987 S2368
S5318	S5266 T175
S5344	S5318 T2987 S2291
S5366	S5344 T1101 S2278 T1263 T801 T801 S2268
S5367	T892 S5366

ID	Use Case 4 Production Rules (SMTC Abstraction)
S6403	T1101 T1105 T892 T1101 T1105 T892 T2987 T801 T921 T248
S6819	T2987 T3243 T252 T248 T1263 T175
S6820	S6403 S6819
S6924	S6820 T248
S6976	S6924 T801
S6977	T0
S7004	T692 S6977 T826 T647
S7005	S6976 S7004
S7007	T892 S7005
S7009	T914 T914
S7012	T248 S7009 T1101 T1105 T892 T1335 T914 T2987 T815 T2987
S7014	T0
S7017	T248 T1101 T1105 T892 T692 T449 T797 T692 T797 T261 T797
S7022	T815 S7014 T801 T1101 T1105 T892 T2987 T248
S7023	S7012 S7022
S7029	S7017 T1101 T1105 T892 T1101 T1105 T892 T797 T2987
S7030	S7029
S7034	T1105 T892 T3243 T815 T1101 T1105 T892
S7035	S7023 T1101 S7034
S7041	T1105 T892 T1101 T1105 T892
S7042	S7030 T1101 S7041
S7044	T1101 T1105 T892
S7045	T2987 S7044 T801 T921 T248

ID	Use Case 4 Production Rules (SMTC Abstraction)
S7048	T801 T1429 T175 T1101 T1105 T892 T1101
S7049	S7035 S7048
S7055	S7042 S7045
S7057	T692 T253
S7063	S7049 T2987 T801 T1430 T431 T1431
S7066	S7063
S7070	T692 S7057 T1263 T175 T924 T815 T2987 T924 T467 T2987 T815
S7071	S7055 T2987 T3243 S7070
S7073	T0
S7081	T1101 S7073 T2987 T801 T2987 T914 T914 T801 T1380 T1223 T801 T871 T1223 T801 T1223 T1223 T1223 T801 T914 T801 T692 T2987 T1101 T1263 T2987 T801 T815 T2987 T801 T2987 T2987 T3379 T2987 T1263 T801 T797 T914 T5593 T2987 T1101 T2987 T914 T2987 T1101 T2987
S7082	S7071 S7081
S7086	T1101 T1263 T1380 T1223 T826 T801 T1223 T801 T1223 T801 T801
S7087	S7066 T1101 T815 T2987 T253 T801 T692 S7086
S7088	S7087
S7098	S7088
S7102 (a)	S2476
S7102 (b)	S5367
S7102 (c)	S7007
S7102 (d)	S7082
S7102 (e)	S7098
S7103	T0 S2250 T1105 S7102 T0

2.5 Use Case 5 - Mozilla Firefox in Windows 10

Use Case 5 is centered around different ways of using Web Browsing. The first scenario (A) simply visits a website. The second and third scenario (B - C) visit a website and download content. Finally, the remaining scenarios (D - F) perform different applications including Facebook Instant Messaging, Twitter and Instagram.

Term/String	Description
T7	Modify File ^.*\Users/user\AppData\Local\.*\$
T58	Modify Registry Key ^.*\$
T60	Modify Registry Key ^.*\Windows\.*\$
T172	Change File ^.*\$
T217	Change File ^.*\Windows\.*\$
T221	Change File ^.*\$
T247	Change File ^.*\Users/user\AppData\Local\.*\$
T251	Change File ^.*\Users/user\AppData\.*\Microsoft\.*\$
T269	Change File ^.*\Program Files (x86)\.*\$
T333	Change File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
T352	Mozilla Firefox Visit URL ^.*\$
S402	Create, Modify, Access, Change MetaInfo, Create and Modify ^.*\Windows\SoftwareDistribution\Download\.*\$
S496	Create File ^.*\Windows\.*\$
S533	Modify, Access and Change MetaInfo File ^.*\Windows\.*\$
S545	Modify, Access and Change MetaInfo File ^.*\$
S547	Create, Create, Create, Create, Create, Create File ^.*\$
S550	Create, Access, Create and Access File ^.*\Users/user\AppData\Local\Microsoft\.*\$
S567	Modify and Change MetaInfo File ^.*\Users/user\AppData\Local\Microsoft\.*\$
S576	Create, Access, Create and Access File ^.*\Program Files (x86)\.*\$
S593	Modify and Access File ^.*\Program Files\WindowsPowerShell\Modules\.*\$
S598	Create, Access, Create, Modify, Access and Change MetaInfo File ^.*\Users/user\AppData\Local\.*\$

Term/String	Description
S603	Access and Change MetaInfo File ^.*\ProgramData\Microsoft\.*\$
S643	Create, Access, Create and Access File ^.*\Users\user\AppData\Local\.*\$
S673	Modify, Change MetaInfo, Modify and Change MetaInfo File ^.*\Users\user\AppData\Local\Microsoft\.*\$
S689	Modify and Change MetaInfo File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S696	Create and Create File ^.*\$
S699	Create, Modify, Access, Change MetaInfo, Create, Access File ^.*\Users\user\AppData\Local\.*\$
S719	Create, Modify, Access, Change MetaInfo, Create, Modify File ^.*\ProgramData\Microsoft\.*\$
S749	Modify and Change MetaInfo File ^.*\Windows\Prefetch\.*\$
S756	Create, Create, Create and Create File ^.*\$
S758	Create and Create File ^.*\Users\user\AppData\Local\.*\$
S764	Modify, Access, Change MetaInfo, Modify, Access and Change MetaInfo File ^.*\Program Files (x86)\Mozilla\Firefox\.*\$
S777	Access and Access File ^.*\Program Files (x86)\.*\$
S785	Create, Modify, Access, Change MetaInfo, Create and Modify File ^.*\Windows\Prefetch\.*\$
S809	Modify and Change MetaInfo File ^.*\Windows\.*\$
S820	Create, Access, Create, Access, Modify and Change MetaInfo File ^.*\$
S832	Modify and Change MetaInfo File ^.*\$
S845	Create, Modify, Access, Change MetaInfo, Create and Modify ^.*\Users\user\AppData\Local\Temp\.*\$

Term/String	Description
S846	Modify, Access, Change MetaInfo, Modify, Access and Change MetaInfo File ^.*\$
S855	Modify and Change MetaInfo File ^.*\ProgramData\Microsoft\.*\$
S872	Modify, Access and Change MetaInfo File ^.*\ProgramData\Microsoft\.*\$
S874	Access, Modify, Access and Change File ^.*\Users\user\AppData\Local\.*\$
S885	Create, Modify, Access, Change MetaInfo, Create and Modify File ^.*\$
S886	Access and Change File ^.*\$
S898	Modify, Access and Change MetaInfo, Modify, Access and Change MetaInfo File ^.*\Windows\.*\$
S904	Create, Modify, Access, Change MetaInfo, Create and Access File ^.*\Program Files (x86)\Mozilla\Firefox\.*\$
S911	Create, Modify, Access and Change MetaInfo File ^.*\Users\user\AppData\Local\.*\$
S922	Access and Change MetaInfo File ^.*\Users\user\AppData\Local\.*\$
S927	Modify and Change MetaInfo File ^.*\Windows\Logs\.*\$
S940	Access and Change MetaInfo File ^.*\Users\user\AppData\Local\Temp\.*\$

Term/String	Description
S946	Modify and Change MetaInfo File ^.*\Users/user\AppData\Local\.*\$
S990	Create, Modify, Access, Change MetaInfo, Create and Modify File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S991	Access and Change MetaInfo File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S994	Create, Modify, Access, Change MetaInfo, Create and Access File
S1001	Modify and Change File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
S1017	Create, Modify, Access, and Change MetaInfo File ^.*\Windows\WinSxS\Manifests\.*\$
S1058	Modify, Access and Change MetaInfo File ^.*\Users/user\AppData\Local\.*\$
S1095	Create, Access, Change MetaInfo, Create, Modify, and Access File ^.*\$
S1102	Create, Access, Create, Modify, Access and Change MetaInfo File ^.*\Program Files (x86)\.*\$
S1130	Modify, Access, Modify, Access and Change MetaInfo File ^.*\Windows\.*\$
S1170	Modify, Access and Change File ^.*\Program Files (x86)\.*\$
S1175	Access and Change MetaInfo File ^.*\Windows\WinSxS\.*\$
S1211	Access and Change MetaInfo File ^.*\Windows\SoftwareDistribution\Download\.*\$
S1218	Access and Change MetaInfo File ^.*\Program Files\WindowsApps\.*\$

Term/String	Description
S1242	Modify, Access, Modify and Access File ^.*\Users/user/AppData/Local/.*\$
S1248	Modify, Change MetaInfo, Modify, Change MetaInfo File ^.*\Users/user/Appdata/Local/.*\$
S1249	Change MetaInfo and Change MetaInfo File ^.*\Users/user/AppData/Local/.*\$
S1252	Modify, Access, Change MetaInfo, Modify, Access, and Change MetaInfo File ^.*\Users/user/AppData/Local/.*\$
S1257	Create, Modify, Access and Change MetaInfo File ^.*\Program Files (x86)/Mozilla Firefox/.*\$
S1258	Modify, Access and Change MetaInfo File ^.*\Program Files (x86)/Mozilla Firefox/.*\$
S1259	Create, Modify, Access, Change MetaInfo, Create and Modify File ^.*\Program Files (x86)/Mozilla Firefox/.*\$
S1260	Access and Change MetaInfo File ^.*\Program Files (x86)/Mozilla Firefox/.*\$
S1261	Modify and Change MetaInfo File ^.*\Program Files (x86)/.*\$
S1263	Modify and Change MetaInfo File ^.*\Users/user/AppData/Local/.*\$
S1269	Create, Access, Create, Access, and Change MetaInfo File ^.*\Windows/SoftwareDistribution/Download/.*\$
S1279	Modify, Access and Change MetaInfo File ^.*\ProgramData/Microsoft/Windows/AppRepository/.*\$
S1295	Access and Change MetaInfo File ^.*\Windows/Prefetch/.*\$

Term/String	Description
S1306	Create, Modify, Access, Change MetaInfo, Create and Access ^.*\Windows\WinSxS\.*\$
S1392	Modify, Change MetaInfo, Modify and Change MetaInfo File ^.*\Program Files (x86)\.*\$
S1399	Access, Modify, Access and Change MetaInfo File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
S1400	Create and Change MetaInfo File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
T1480	Access File ^.*\Windows\.*\en-US\.*\$
S1491	Access File ^.*\Program Files\WindowsPowerShell\Modules\.*\$
S1492	Change File ^.*\$
S1525	Change File ^.*\$
S2910	S402 and S1211 ^.*\Windows\SoftwareDistribution\Download\.*\$
S2960	S1077 and S1218 File ^.*\Program Files\WindowsApps\.*\$
S2999	S845 and S940 File ^.*\Users\user\AppData\Local\Temp\.*\$
S3015	S567 and S922 File ^.*\Users\user\AppData\Local\.*\$
S3055	S990 and S991 File ^.*\ProgramData\Microsoft\Windows\AppRepository\.*\$
S3107	S719 and S603 File ^.*\ProgramData\Microsoft\.*\$
S3110	S885 and S886 File ^.*\$
S3391	S953 and S1175 File ^.*\Windows\WinSxS\.*\$
S3447	S1259 and S1260 File ^.*\$
S4765	Change File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
S4893	S3447 and S3015 File ^.*\Program Files (x86)\Mozilla Firefox\.*\$
S4915	S3110 and S2910 File ^.*\$
S5341	S3015 and S1269 File ^.*\Users\user\AppData\Local\.*\$
S5444	S3015 and S3391 File ¹⁵⁹ ^.*\Users\user\AppData\Local\.*\$
S5448	S1306 and S3015 File ^.*\Windows\WinSxS\.*\$
S5479	S3015 and S2910 File ^.*\Users\user\AppData\Local\.*\$

Scenario	Sequence of Terms and String for Mozilla Firefox (DVSMT Abstraction)
UC5A	749 58 832 832 749 1491 1525 922 598 946 545 820 221 832 545 1252 832 749 545 58 1058 545 885 886 885 886 1480 545 545 1252 1058 874 1252 946 1252 946 946
UC5B	1279 749 832 946 749 832 696 885 886 846 885 886 885 886 994 221 832 545 1480 1480 946 576 1170 990 991 1170 1258 1170 699 545 885 886 885 886 1480 533 749 946 1170 1392 251 1491 1492 832 1077 1218 1491 922 946 1252 1480 885 886 885 886 1480
UC5C	58 749 777 269 749 1257 1392 269 576 1399 333 1001 764 1399 764 749 764 1001 1399 764 1261 1399 764 1001 1252 1170 1258 1001 1258 1259 1260 1258 1259 1260 1258 1400 1170 1259 1260 58 1170 1392 1102 872 719 603 172 1170 911 1058 845 940 545 885 886 832 533 749 785 1295 58 1259 1260 832 885 886 885 886 820 749 885 886 885 886 904 333 904 333 1259 1260 1480 221 1095 221 598 352 1259 1260 1259 1260 1259 1260 1491 922 946 1491 1525 1492 598 1491 922 946 1491 922 598 946 1480 946 1491 922 598 946 1480 946 545 885 886 545 545 832 545 689 809 1480 1058 749 749 809 545 885 886 872 719 603 567 885 886 1480 1058 598 749 496 1058 1480
UC5E	58 749 832 946 221 820 545 832 545 1058 1252 832 545 1480 545 885 886 1480 885 886 749 496 1058

Scenario	Sequence of Terms and String for Mozilla Firefox (DVSMT Abstraction)
UC5F	58 58 550 673 855 946 58 749 855 832 749 1130 885 886 60 832 58 927 885 886 994 749 785 1295 749 533 785 1295 172 60 58 533 217 898 809 749 696 946 7 946 247 946 885 886 994 1480 545 885 886 846 885 886 598 1491 1525 922 885 886 885 886 758 598 1491 922 598 643 1263 946 1480 946 247 643 1242 946 1248 1480 1058 1480 1249 946 545 221 547 756 598 699 946 885 886 885 886 545

Scenario	Sequence of Terms and String for Mozilla Firefox (SMTC Abstraction)
UC5A	749 58 832 832 749 3015 598 946 545 820 221 832 545 1252 832 749 545 58 1058 545 3110 3110 3015 545 545 1252 1058 874 1252 946 1252 946 946
UC5B	1279 749 832 946 749 832 696 3110 846 3110 994 221 832 545 5444 5448 946 576 1170 3055 1170 1258 1170 699 545 3110 3110 3015 533 749 946 1170 1392 251 3015 832 2960 3015 946 1252 3015 3110 3015
UC5C	58 749 777 269 749 1257 1392 269 576 1399 333 1001 764 1399 764 749 764 1001 1399 764 1261 1399 764 1001 1252 1170 1258 1001 1258 3447 1258 3447 1258 1400 1170 3447 58 1170 1392 1102 872 3107 172 1170 911 1058 2999 545 832 533 749 785 1295 58 3447 832 4915 3110 820 749 3110 3110 904 333 904 333 4893 221 1095 221 598 352 3447 3015 946 3015 598 3015 946 3015 598 946 3015 946 3015 598 946 3015 946 545 3110 545 545 832 545 689 809 3015 1058 749 749 809 545 3110 872 3107 567 3110 3015 1058 598 749 496 1058 5341
UC5E	58 749 832 946 221 820 545 832 545 1058 1252 832 545 5479 545 3110 4765 749 496 1058

Scenario	Sequence of Terms and String for Mozilla Firefox (SMTC Abstraction)
UC5F	58 58 550 673 855 946 58 749 855 832 749 1130 3110 60 832 58 927 3110 994 749 785 1295 749 533 785 1295 172 60 58 533 217 898 809 749 696 946 7 946 247 946 3110 994 3015 545 3110 846 3110 598 3015 3110 758 598 3015 598 643 1263 946 3015 946 247 643 1242 946 1248 3015 1058 3015 1249 946 545 221 547 756 598 699 946 3110 3110 545

ID	Use Case 5 Production Rules (DVSMT Abstraction)
S3	T1491 T1525 T922 T598 T946 T545 T820
S4	T832 T545 T1252
S5	S3 T221 S4 T832 T749
S6	T58 T1058 T545 T885 T886 T885 T886
S7	T1252 T1058 T874 T1252 T946 T1252 T946
S8	S5 T545 S6 T1480 T545 T545 S7
S9	T1170 T1392 T251 T1491 T1492 T832 T1077 T1218 T1491 T922 T946 T1252 T1480 T885 T886 T885 T886 T1480
S13	S8 T946
S16	T885 T886 T885 T886 T1480 T533 T749
S24	T946 T576 T1170 T990 T991 T1170 T1258 T1170 T699
S238	T832 T696 T885 T886 T846 T885 T886 T885 T886 T994
S410	S238 T221 T832
S460	S410 T545 T1480 T1480 S24 T545 S16
S461	S460 T946 S9
S525	T1257 T1392 T269 T576 T1399 T333 T1001 T764 T1399 T764 T749 T764 T1001 T1399 T764 T1261 T1399 T764 T1001

ID	Use Case 5 Production Rules (DVSMT Abstraction)
S535	T1252 T1170 T1258 T1001 T1258 T1259 T1260 T1258 T1259 T1260 T1258 T1400 T1170 T1259 T1260 T58 T1170 T1392 T1102 T872 T719 T603 T172 T1170 T911 T1058 T845 T940 T545 T885 T886 T832 T533 T749 T785 T1295 T58 T1259 T1260 T832 T885 T886 T885 T886 T820 T749 T885 T886 T885 T886 T904 T333 T904 T333 T1259 T1260 T1480 T221 T1095 T221 T598 T352 T1259 T1260 T1259 T1260 T1259 T1260 T1491 T922 T946 T1491 T1525 T1492 T598 T1491 T922 T946 T1491 T922 T598 T946 T1480 T946 T1491 T922 T598 T946 T1480 T946 T545 T885 T886 T545 T545 T832 T545 T689 T809 T1480 T1058 T749 T749 T809 T545 T885 T886 T872 T719 T603 T567 T885 T886 T1480 T1058 T598 T749 T496 T1058 T1480
S536	S525 S535
S835	T1279 T749
S1131	S835 T832 T946
S1384	S1131 T749 S461
S1426	T749 T58
S1427	S1426 T832 T832
S1430	S1427 T749 S13
S2607	T58 T749
S2609	S2607 T832
S2611	T58 T58 T550 T673 T855
S2616	T946 T58 T749 T855 T832 T749 T1130 T885 T886 T60
S2619	S2616 T832
S2621	T927

ID	Use Case 5 Production Rules (DVSMT Abstraction)
S2625	T58 S2621 T885 T886 T994 T749 T785 T1295 T749 T533 T785 T1295 T172 T60 T58 T533 T217 T898 T809 T749 T696
S2626	S2619 S2625
S2628	T885 T886 T994
S2629	T545 T885 T886 T846
S3224	T58 T749 T777 T269
S3519	S3224 T749 S536
S3521	T221 T820 T545 T832
S3523	T946 S3521 T545 T1058 T1252
S3526	S3523 T832 T545 T1480
S3529	T885 T886 T749 T496
S3538	S3526 T545 T885 T886 T1480 S3529 T1058
S3539	S3538
S3540	S3539
S3543	S2609 S3540
S3545	T946 S2628 T1480 S2629 T885 T886 T598 T1491 T1525 T922 T885 T886 T885 T886 T758 T598 T1491 T922 T598 T643 T1263 T946 T1480 T946 T247 T643 T1242 T946 T1248 T1480 T1058 T1480 T1249 T946 T545 T221 T547 T756 T598 T699 T946 T885 T886 T885 T886 T545
S3546	S2626 T946 T7 T946 T247 S3545

ID	Use Case 5 Production Rules (DVSMT Abstraction)
S3547	S3546
S3549	S2611 S3547
S3552 (a)	S1384
S3552 (b)	S1430
S3552 (c)	S3519
S3552 (d)	S3543
S3552 (e)	S3549
S3553	T0 S3552 T0

ID	Use Case 5 Production Rules (SMTC Abstraction)

S3	T3015 T598 T946 T545 T820
S4	T1252 T832 T749 T545 T58 T1058 T545 T3110 T3110 T3015 T545
S5	S3 T221 T832 T545 S4
S6	T1252 T1058 T874 T1252
S7	S5 T545 S6
S8	T1252 T946 T946
S9	S7 T946 S8
S12	T1170 T1392 T251 T3015 T832 T2960 T3015 T946 T1252 T3015 T3110 T3015
S16	T3110 T3110 T3015 T533 T749
S24	T5444 T5448 T946 T576 T1170 T3055 T1170 T1258 T1170 T699
S40	T832 T696 T3110 T846 T3110 T994
S56	S40 T221 T832 T545 S24
S64	S56 T545 S16
S68	S64 T946 S12
S87	T1257 T1392 T269 T576 T1399 T333 T1001 T764 T1399 T764 T749 T764 T1001 T1399 T764 T1261 T1399 T764 T1001
S96	S87 T1252

ID	Use Case 5 Production Rules (SMTC Abstraction)

S100	T1170 T1258 T1001 T1258 T3447 T1258 T3447 T1258 T1400 T1170 T3447 T58 T1170 T1392 T1102 T872 T3107 T172 T1170 T911 T1058 T2999 T545 T832 T533 T749 T785 T1295 T58 T3447 T832 T4915 T3110 T820 T749 T3110 T3110 T904 T333 T904 T333 T4893 T221 T1095 T221 T598 T352 T3447 T3015 T946 T3015 T598 T3015 T946 T3015 T598 T946 T3015 T946 T3015 T598 T946 T3015 T946 T545 T3110 T545 T545 T832 T545 T689 T809 T3015 T1058 T749 T749 T809 T545 T3110 T872 T3107 T567 T3110 T3015 T1058 T598 T749 T496 T1058 T5341
S101	S96 S100
S129	T1279 T749
S155	S129 T832 T946
S171	S155 T749 S68
S180	T749 T58
S181	S180 T832 T832
S182	S181 T749 S9
S281	T58 T749
S285	T58 T58 T550 T673 T855
S288	S285
S291	T946 T58 T749 T855
S292	T0
S293	T927

ID	Use Case 5 Production Rules (SMTC Abstraction)
S299	T832 S292 T58 S293 T3110 T994 T749 T785 T1295 T749 T533 T785 T1295 T172 T60 T58 T533 T217 T898 T809 T749 T696
S300	S291 T832 T749 T1130 T3110 T60 S299
S301	S300
S304	T946 T7 T946 T247 T946 T3110 T994 T3015 T545 T3110 T846 T3110 T598 T3015 T3110 T758 T598 T3015 T598 T643 T1263 T946 T3015 T946 T247 T643 T1242 T946 T1248 T3015 T1058 T3015 T1249 T946 T545 T221 T547 T756 T598 T699 T946 T3110 T3110 T545
S305	S301 S304
S308	S288 S305
S358	T58 T749 T777 T269
S384	S358 T749 S101
S385	T832 T946 T221 T820 T545
S386	S281 S385
S388	T832 T545 T1058 T1252
S389	T3110 T4765 T749 T496
S398	T545 S389 T1058
S399	S388 T832 T545 T5479 S398
S400	S399

ID	Use Case 5 Production Rules (DVSMT Abstraction)
S405	S386 S400
S406 (a)	S171
S406 (b)	S182
S406 (c)	S308
S406 (d)	S384
S406 (e)	S405
S407	T0 S406 T0

Appendix C. Parameter File

```
////////////////////////////////////////
// This is a template for the TEARCPP Parameter File //
//      All possible parameters are included          //
//      booleans use 1 for true, 0 for false          //
////////////////////////////////////////

// Debug information
//      debug - level of detail to report (0 is least)
//      logFilename - file to print debug information (stdout if nothing specified)
//      increment - for reporting to screen, number of records to process before next
Debug,5
LogFilename,"log.txt"
Increment,1000

// Level of abstraction to use as input
//      for abstraction, output level is 5 more than existing level
//      So, to abstract a level, "-abstract" will abstract using strings to level + 5
//      Then, "-loops" will create loops to level + 5
//      e.g. -abstract 10,   -loops 15
ExistingLevel,10

// Case numbers and Date Range to use as input
//      If date range is the same, use CaseNbr, StartDate, End Date
//      If date ranges vary, use dateranges
//      Do not use both
//      NB: do not use leading zeros -- turns input into octal
CaseNbr,"UC1a,uc1b,uc1c,uc1d,uc1e,uc1f,uc1g,uc1h"
StartDate,1012001
StartTime,1
EndDate,12312099
EndTime,235959
DateRange,UC1a,1012001,1,12312099,235959

// Use Item Pattern Lists
//      -- significant upfront overhead, but speeds processing of large jobs
UseItemPatternLists,1
```

```

// Report Parameters:
//   outFilename - file to send report to (only relevant for -events,
//               -strings, -search)
//   items - show item details (only relevant for -events, -search)
//   maxItem - maximum number of items to show when showing item detail (0
//               for all items) (only relevant for -events, -search)
//   details - show string level details (i.e., terms that make up a string)
//               (only relevant for -events, -strings, -search)
//   counts - show string counts (only relevant for -strings)
//   zeros - show strings that have no occurrences (only relevant for
//               -strings); NB -- if zeros aren't displayed, strings
//               that only occur embedded in other strings are not printed
//   terms - show terms (only relevant for -strings)
//   strings - show strings (only relevant for -strings)
//   timespan - range in seconds for two activities to be part of same string
//               (only relevant for -search)
//   bufferTime - buffer around first and last event to also include (only
//               relevant for -search)
outFilename,"output.txt"
items,1
maxItem,5
details,1
counts,0
zeros,1
terms,1
strings,1
timespan,60

// Grammar Build/ Abstraction Parameters
//   maximumStringLength - largest string/ string instance that will be
//               created
//   thresholdSimilar - maximum Levenshtein edit distance for strings to be
//               considered variations of each other
//   showValues - show the "score" for all strings (otherwise, only strings
//               with scores above threshold are shown)
//               (only relevant for -dvsmt, -smtc)
//   bufferTime - maximum number of seconds between first and last term to
//               be considered connected
//               (only relevant for -sosla, -smtc, -abstract, -loops)
//   epsilon - weight for length, inverse weight for generic used (only
//               relevant for -smtc)

```

```

//      thresholdScore - minimum score for a string to be included (only relevant
//                          for -smtc)
//      thresholdMain - minimum score for a string to be included (only relevant
//                          for -dvsmt)
//      thresholdVariation - minimum score for a variation of a main string to be
//                          included (only relevant for -dvsmt)
//      multiItem - allow strings with multiple items to be generated (only
//                          relevant for -sosla)
//      singleString - create one string that encompasses entire date range
//                      (overrides maximum string length) (only relevant for -sosla)
//      partialMatch - abstract activities if partially matched using threshold
//                      similarity (only relevant for -abstract)
MaximumStringLength,6
ThresholdSimilar,3
showValues,1
BufferTime,30
Epsilon,3.0
ThresholdScore,1
ThresholdVariation,1.5
ThresholdMain,1.1
MultiItem,0
SingleString,0
PartialMatch,0

// Data Extraction Parameters
//      path - path of input files; expects each case will have a folder in the
//              path directory (uses that folder name to create CaseNbr)
//      gmtoffset - offset in seconds from GMT
Path,"c:/users/user/desktop/dissertation work/useCases/usecase5/"
GMTOffset,0

// Exclusions - only used in ConsolidateStrings
//      EXCLUSIONS -- designates all subsequent lines as timelines, string ids,
//                      or term ids, to be excluded
//      NB: any recognized keyword from this parameter file terminates the
//          EXCLUSIONS blocks
EXCLUSIONS
1079
29130797
639

// Consolidate String Parameters

```

```

//      stringName -- name to associate with the consolidated string
//      generateString -- create the string records in the current database
stringName,"StartMicrosoftWord"
generateString,1

// Search Parameters
//      INSTANCES -- designates that all subsequent lines are patterns for items
//                  to match
//                  -- these patterns are of the form InstanceId,ItemType,Pattern
//                  -- itemtype is 5 for items (only type currently implemented)
//                  -- pattern is in double quotes and is a regular expression
//      SEARCHES -- designates that all subsequent lines are searches
//                  -- these searches are of the form SearchId,TermId,InstanceID
//                  -- instanceID is described above
//                  -- termID is the id of the term or string found by printing a
//                  strings report (i.e., -strings)
//                  -- searchID links multiple lines together; the order of the
//                  terms is determined by the order they appear
//      NB: any recognized keyword from this parameter file terminates the INSTANCES
INSTANCES
50,5,".*prefetch/winword.*"
SEARCHES
105,1,50
105,2,50
105,40,50
105,202,50
110,1,50
110,40,50
115,2,50
115,202,50

```

Appendix D. Algorithms

4.1 Single Object String and Loop Abstraction (SOSLA)

Header.

```
#pragma once
#include <unordered_map>
#include "Parameters.h"
#include "Terms.h"
#include "Generics.h"
#include "Timeline.h"
#include "TimelineEntry.h"
#include "Strings.h"
#include "Patterns.h"
#include "ActivityBag.h"
#include "queues.h"
#include "utilities.h"
#include "ItemUtilities.h"
#define htActivityBag_i std::unordered_map<uint32_t, ActivityBag *>::const_iterator

class Sosla
{
    ItemUtilities *itemUtilities;
    Utilities *utilities;
    Parameters *parameters;
    Generics *generics;
    Patterns *patterns;
    Strings *strings;
    Terms *terms;
    Timeline *timeline;
    std::unordered_map<uint32_t, ActivityBag *> *htActivityBags;

public:
    Sosla::Sosla(Utilities *wutilities, ItemUtilities *witemUtilities, Parameters
                *wparameters, Patterns *wpatterns, Strings *wstrings, Generics
                *wgenerics, Terms *wterms);
    ~Sosla();
    void Sosla::process();
};
```

C++.

[illegible]

```
Sosla::Sosla(Utilities *wutilities, ItemUtilities *witemUtilities, Parameters
             *wparameters, Patterns *wpatterns, Strings *wstrings, Generics
             *wgenerics, Terms *wterms) {
    itemUtilities = witemUtilities;
    utilities = wutilities;
    parameters = wparameters;
    patterns = wpatterns;
    strings = wstrings;
    terms = wterms;
    generics = wgenerics;
    htActivityBags = new std::unordered_map<uint32_t, ActivityBag *>;
}
```

```
Sosla::~~Sosla() {}
```

```
void Sosla::process() {
    ActivityBag abag;                // Generic activity bag to give access to
                                     // ActivityBag methods
    queueT<TimelineEntry> *qhead;    // The list of timeline entries to use to
                                     // create an activity bag

    TimelineEntry *entry;            // generic timeline entry variable

    uint8_t isFirst;                 // True if this is the first iteration
    uint8_t keepGoing;               // False if it's time to create strings

    uint32_t i;                      // generic loop variable
    uint32_t curEndTime;              // the end time of the first record in the
```

```

uint32_t oldItem;           // workspace
                             // the current item being matched (for single
                             // strings)
uint32_t wlength;          // the size of the workspace
uint32_t saveLength;       // the size of the workspace (used for
                             // multi-string to look at all smaller sized
                             // workspaces)

uint32_t *orderList;       // states that the activities sent to
                             // addActivityBag are in the right order

// Reduces workspace after processing for multi-item SOSLA
TimelineEntry *entry2;
uint32_t ctr2;

////////////////////////////////////

// Initialize Variables
qhead = new queueT<TimelineEntry>;

orderList = (uint32_t *)malloc(sizeof(uint32_t) * parameters->maxString);
for (i = 0; i < parameters->maxString; i++) orderList[i] = i;

// Initialize processing of event records
timeline = new Timeline(utilities, parameters, terms,generics);
if (timeline->initialize() == NULL) return;

do { // for each case
    // Iterate through the entire dataset
    isFirst = 1;
    qhead->qclear();
    while ((entry = timeline->process(1)) != NULL) {

        // First iteration -- just get things started
        if (isFirst) {
            isFirst = 0;
            qhead->qpush(entry);
            curEndTime = entry->getEndTime();
            oldItem = entry->getItems()->getItem();
            wlength = 1;
            continue;
        }
    }
}

```



```

// If we're still within the buffer, keep getting records
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "checking if still in scope\n");
keepGoing = 0;
if (parameters->singleString) keepGoing = 1;
else if (wlength < parameters->maxString && entry->getStartTime() -
        curEndTime < parameters->bufferTime) {
    if (parameters->doMulti) keepGoing = 1;
    else if (itemUtilities->sameAs(entry->getItems()->getItem(),
        oldItem)) keepGoing = 1;
}

// If still adding events, just push the current event into the queue
// and increase the size of the workspace by one
if (keepGoing) {
    qhead->qpush(entry);
    wlength++;
    continue;
}

// Otherwise,
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "***time to add stuff***    wlength = %i\n", wlength);

// if doing multiple-object strings, do every possible string in the
// buffer, otherwise just add one activity bag for the whole workspace
if (parameters->doMulti) saveLength = wlength;
while (1) {
    if (wlength > 1)
        abag.addActivityBag(qhead, orderList, wlength, 0, 0, strings,
            htActivityBags, parameters, utilities, patterns);
    if ((!parameters->doMulti) || (wlength <= 1)) break;
    wlength--;
}
if (parameters->doMulti) wlength = saveLength;

// Get rid of all of the records in buffer that are outside new buffer
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "removing out of scope activities\n");
ctr2 = 0;
while (!qhead->qempty() && wlength > 0) {
    entry2 = qhead->qvalue(qhead->initializeIterator());

```

```

        curEndTime = entry2->getEndTime();

        if (entry2->getStartTime() - curEndTime < parameters->bufferTime) {
            if (parameters->doMulti && ctr2 >= 2) break;
            if (!parameters->doMulti &&
                !itemUtilities->sameAs(entry2->getItems()->getItem(), oldItem))
                break;
        }

        ctr2++;
        entry2 = qhead->qpop();
        delete entry2;
        wlength--;
    }

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "\t %u records removed\n", ctr2);

    // if the workspace is empty, set the end time for the first entry into
    // the workspace to the record we're about to add to the workspace
    if (qhead->qempty()) curEndTime = entry->getEndTime();

    // add the current event to the workspace
    qhead->qpush(entry);
    wlength++;

    // set the item being matched the current one (this only matters if we
    // just emptied the workspace)
    oldItem = entry->getItems()->getItem();

    // if we're already past the end of the records
    if (parameters->singleString) break;
}
} while (timeline->iterate());

// write the strings, generics, and patterns to the database
abag.createString(htActivityBags, strings);
generics->writeGenericItems();
patterns->writePatterns();
}

```

4.2 Discontinuous Varied Order Sequence Mining with Templates (DVSMT)

Header.

```
#pragma once
#include <unordered_map>
#include "Parameters.h"
#include "Terms.h"
#include "Generics.h"
#include "Timeline.h"
#include "TimelineEntry.h"
#include "Strings.h"
#include "Patterns.h"
#include "ItemInstance.h"
#include "queues.h"
#include "Print.h"
#include "ActivityBag.h"
#include "utilities.h"
#include "ItemUtilities.h"

#define htActivityBag_i std::unordered_map<uint32_t, ActivityBag *>::const_iterator
class dvsmt
{
    typedef struct timelineXrefType {
        TimelineEntry *entry;
        uint32_t prior;
        uint32_t next;
    } timelineXref;

    ItemUtilities *itemUtilities;
    Utilities *utilities;
    Parameters *parameters;
    Generics *generics;
    Patterns *patterns;
    Strings *strings;
    Terms *terms;
    Timeline *timeline;
    Print *prtAction;
    ActivityBag *activityBag;

    std::unordered_map<uint32_t, ActivityBag *> **htActivityBags;

public:
    dvsmt(Utilities *utilities, ItemUtilities *itemUtilities, Parameters
```

```

        *wparameters, Patterns *wpatterns, Strings *wstrings, Generics
        *wgenerics, Terms *wterms);
~dvsmt();
void dvsmt::process();
uint32_t dvsmt::keepInteresting(std::unordered_map<uint32_t, ActivityBag *>
                                *htActivityBags, uint32_t datasetLength);
};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Discontinuous Varied order Sequential Mining with Templates (DVSMT) //
// Input: a dataset to generate a set of production rules (terms and strings) //
// Output: a set of production rules                                           //
// Notes:                                                                                               //
// Initial Release: 13-Jul-17                                                  //
//                                                                                               //
// Revision History:                                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "DVSMT.h"

dvsmt::dvsmt(Utilities *wutilities, ItemUtilities *witemUtilities, Parameters
             *wparameters, Patterns *wpatterns, Strings *wstrings, Generics
             *wgenerics, Terms *wterms) {

    itemUtilities = witemUtilities;
    utilities = wutilities;
    parameters = wparameters;
    patterns = wpatterns;
    strings = wstrings;
    terms = wterms;
    generics = wgenerics;
    htActivityBags = (std::unordered_map<uint32_t, ActivityBag *> **)
        malloc(sizeof(std::unordered_map<uint32_t, ActivityBag *>*) *
               parameters->maxString);
    prtAction = new Print(utilities, witemUtilities, parameters, patterns,
                          generics, terms, strings);
    activityBag = new ActivityBag();

}

dvsmt::~dvsmt() {}

```

```

void dvsmt::process() {

    // Activity Bags kept for each string length
    htActivityBag_i htiter;
    std::unordered_map<uint32_t, ActivityBag *> *htAbag;

    // The list of timeline entries to use to create an activity bag
    queueT<TimelineEntry> *qhead;
    void *qtbagIter;

    TimelineEntry *entry;           // generic timeline entry variable
    TimelineEntry *lastEntry;       // generic timeline entry variable - used in
                                    // Size-1 loop to build double linked list for
                                    // entries

    queueT<TimelineEntry> *tbags;   // all of the instantiations for this activity
                                    // bag
    TimelineEntry *firstEntry;      // the first timeline entry for this
                                    // instantiation of the string

    ActivityBag *abag;              // the activity bag being examined

    uint32_t *orderList;            // states that the activities sent to
                                    // addActivityBag are in the right order
    uint32_t datasetLength;         // number of events in the dataset (excludes
                                    // loops as they are skipped in
                                    // timeline->process)
    uint32_t j;                    // generic loop variable

    int ctr;                        // number of activity bags looked at
    uint32_t abagSize;              // number of activity bags retained at each
                                    // level

    uint8_t skipit;                // true if this set of timelines needs to be
                                    // skipped for bad input
    uint32_t wlength;              // current string length being examined

    uint32_t whash;                // the hash of the returned abag

    // Status update variables
    uint32_t ctr2, last_seen;
    time_t wtimer;

```

```

struct tm timeinfo;
char showtime[80];

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Initialize Variables
orderList = (uint32_t *)malloc(sizeof(uint32_t) * parameters->maxString);
for (j = 0; j < parameters->maxString; j++) orderList[j] = j;
htAbag = new std::unordered_map<uint32_t, ActivityBag *>;
qhead = new queueT<TimelineEntry>;
datasetLength = 0;

// Initialize processing of events -- get the first case number
timeline = new Timeline(utilities, parameters, terms, generics);
if (timeline->initialize() == NULL) return;

// Loop through the dataset once
//      1. create a doubly linked list for getting prior/next timeline later
//      2. determine the dataset length for later determination of what strings
//          to keep
//      3. create 1-length strings
htActivityBags[0] = htAbag;
do {
    lastEntry = NULL;
    while ((entry = timeline->process(1)) != NULL) {
        datasetLength++;

        // set prev/next pointers
        if (lastEntry != NULL) lastEntry->setNext(entry);
        entry->setPrev(lastEntry);
        lastEntry = entry;

        // Create the 1-length activity bag for this entry
        qhead->qclear();
        qhead->qpush(entry);
        activityBag->addActivityBag(qhead, orderList, 1, 1, 0,
                                   strings, htActivityBags[0], parameters, utilities, patterns);
    }
    if (lastEntry != NULL) lastEntry->setNext(NULL);
} while (timeline->iterate());

// Keep the interesting abags
time(&wtimer); localtime_s(&timeinfo, &wtimer);

```

```

strftime(showtime, 80, "Current time: %X", &timeinfo);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\tDetermining how many are interesting\t\t%s\n", showtime);
abagSize = keepInteresting(htActivityBags[0], datasetLength);
printf("%u interesting activity bags at level 0\n", abagSize);

// Go through the remaining interesting list and create two new abags that
// are each 1 term bigger
//      1. One with the previous timeline's activity and the timelines from
//      this instantiation
//      2. One with timelines from this instantiation followed by the next
//      timeline's activity
for (wlength = 1; wlength < parameters->maxString; wlength++) {
    if ((htActivityBags[wlength - 1])->empty()) break;

    // Clear out the older activity bags to save space
    if (wlength > 1) {
        for (htiter = (htActivityBags[wlength - 2])->begin();
            htiter != (htActivityBags[wlength - 2])->end(); htiter++) {
            abag = (htiter->second);
            delete abag;
        }
        htActivityBags[wlength - 2]->clear();
    }

    // Create a container for the new activity bags
    htAbag = new std::unordered_map<uint32_t, ActivityBag *>;
    htActivityBags[wlength] = htAbag;

    // For each instantiation of each interesting activity bag of the current
    // size, create two new activity bags
    ctr = 0;
    for (htiter = (htActivityBags[wlength - 1])->begin();
        htiter != (htActivityBags[wlength - 1])->end(); htiter++) {
        abag = (htiter->second);

        // report progress
        ctr++;
        time(&wtimer); localtime_s(&timeinfo, &wtimer);
        strftime(showtime, 80, "Current time: %X", &timeinfo);
        printf("\tReviewing %i of %u activity bags (%u timeline entries)\t\t%s\n",
            ctr, abagSize, abag->getCount(), showtime);
    }
}

```

```

// Iterate through all of the instantiations of the current activity bag
ctr2 = 0; last_seen = 0;
tbags = abag->getTimelines();
if (tbags == NULL || tbags->qempty()) {
    printf("no timelines for this activity bag -- skipping \n");
    continue;
}
for (qtbagIter = tbags->initializeIterator(); qtbagIter != NULL;
     qtbagIter = tbags->iterate(qtbagIter)) {
    if (ctr2 > abag->getCount()) break;
    if ((firstEntry = tbags->qvalue(qtbagIter)) == NULL) continue;

    // Report progress
    if (utilities->showTime(0, &ctr2, &last_seen,
        parameters->increment))
        printf("\t\tTermcount = %i\t\tActivity Bag Count = %u\n",
            (int)parameters->termCount, parameters->abagCount);

    // Clear the old list
    qhead->qclear();
    skipit = 0;

    // Build the first new activity bag

    // First entry (prior to the current abag's timelines)
    if (firstEntry->getPrev() != NULL)
        qhead->qpush(firstEntry->getPrev());
    else skipit = 1;

    // The rest of the entries (the current abag's timelines)
    entry = firstEntry;
    if (!skipit) for (j = 0; j < wlength; j++) {
        qhead->qpush(entry);
        entry = entry->getNext();
        if (entry == NULL) {skipit = 1; break;}
    }

    if (skipit == 0) {

        // Add the first activitybag
        whash = activityBag->addActivityBag(qhead, orderList,
            wlength + 1, 1, 0, strings, (htActivityBags[wlength]),

```



```

        parameters, utilities, patterns);
abag = (*(htActivityBags[wlength]))[whash];
abag->checkVariations(htActivityBags[wlength],
        parameters->thresholdSimilar, utilities);

// Build the second activity bag
// Remove the first entry from the queue (the prior timeline
// added above)
qhead->qpop();

// Add the timeline after the current one (where entry points
// after the above For loop finishes)
qhead->qpush(entry);

// Add the second activity bag
whash = activityBag->addActivityBag(qhead, orderList,
        wlength + 1, 1, 0, strings, (htActivityBags[wlength]),
        parameters, utilities, patterns);
abag = (*(htActivityBags[wlength]))[whash];
abag->checkVariations(htActivityBags[wlength],
        parameters->thresholdSimilar, utilities);
    }

}

// keep the interesting abags
time(&wtimer); localtime_s(&timeinfo, &wtimer);
strftime(showtime, 80, "Current time: %X", &timeinfo);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\tDetermining how many are interesting\t\t\t%s\n", showtime);
abagSize = keepInteresting(htActivityBags[wlength], datasetLength);
printf("%u interesting activity bags at level %i\n", abagSize, wlength);

// Create the strings
activityBag->createString(htActivityBags[wlength], strings);
}

// Add any generics and patterns added during processing to the database
generics->writeGenericItems();
patterns->writePatterns();
strings->WriteStrings();

```

```

}

uint32_t dvsmt::keepInteresting(std::unordered_map<uint32_t, ActivityBag *>
                               *htActivityBags, uint32_t datasetLength) {
    // Purpose: determine which activity bags meet the threshold for being
    //           "interesting"

    ActivityBag *abag, *mainAbag;    // the activity bag holding the term being
                                     // considered
    TermList *wterm;                 // individual terms within the string being
                                     // considered
    uint8_t keepit;                  // true if we decide to keep this activity bag
    double measure;                  // the score of the string being considered
    uint32_t wcount;                 // the number of strings that meet the
                                     // threshold

    // Iterators and Counters
    htActivityBag_i htiter, witer;
    int rc;                          // result code

    //////////////////////////////////////

    // Initialize Variables
    wcount = 0;

    // Iterate through all the activity bags
    for (htiter = htActivityBags->begin(); htiter != htActivityBags->end();
         htiter++) {
        abag = htiter->second;

        // If they want to see the scores, start by showing them the string being
        // considered
        if (parameters->showValues) {
            printf("main      %llx\n", (uint64_t)(abag->getMainActivityBag()));
            wterm = abag->getActivities();
            rc = prtAction->prtString(wterm, 0);
            if (rc) while (wterm != NULL) {
                if (wterm->getTerm() <= 0 || wterm->getTerm() > terms->getMaxTerm())
                    break;
                prtAction->prtTerm(1, wterm->getTerm(), 0);
                wterm = wterm->getNext();
            }
        }
    }
}

```

```

}

// Check if this is a variant
mainAbag = abag->getMainActivityBag();
if (mainAbag == NULL) { // no -- it's the main (and possibly only)
    // sequence for this string

    // Calculate its score and whether it passes threshold
    measure = ((double)datasetLength) / ((double)(abag->getLength() +
        (datasetLength - abag->getLength() * abag->getCount())));
    keepit = (measure > parameters->thresholdMain);

    // if they want to see the scores, show them
    if (parameters->showValues) {
        fprintf(parameters->logFile,
            "\t dataSet length = %i      \t string length = %i    count = %i\n",
            datasetLength, abag->getLength(), abag->getCount());
        fprintf(parameters->logFile,
            "\t measure1 = %f threshold = %f    keepit = %u\n",
            measure, parameters->thresholdMain, keepit);
    }
}

else { // yes, it's a variant of a main sequence

    // if we're showing things, show the main sequences length and
    // frequency (but (right now) we're not showing the main sequence)
    if (parameters->showValues)
        fprintf(parameters->logFile,
            "\t**main ptr**\n\t\t main ptr:    length = %u        count = %u\n",
            mainAbag->getLength(), mainAbag->getCount());

    // Calculate its score and whether it passes threshold
    measure = ((double)(datasetLength - (mainAbag->getLength() *
        mainAbag->getCount())) /
        ((double)(datasetLength - (abag->getLength() * abag->getCount())));
    keepit = (measure > parameters->thresholdVariation);

    // if they want to see the scores, show them
    if (parameters->showValues) {
        wterm = mainAbag->getActivities();
        rc = prtAction->prtString(wterm, 0);
        if (rc) while (wterm != NULL) {
            prtAction->prtTerm(1, wterm->getTerm(), 0);
        }
    }
}

```

```

        wterm = wterm->getNext();
    }
    fprintf(parameters->logFile,
        "\t**main ptr**\n\tstring length = %i    count = %i\n",
        abag->getLength(), abag->getCount());
    fprintf(parameters->logFile,
        "\tmeasure2 = %f threshold = %f    keepit = %u\n",
        measure, parameters->thresholdVariation, keepit);
    }
}

// if we're showing things, make space between this and the next string
// under consideration
if (parameters->showValues) fprintf(parameters->logFile, "\n\n");

if (!keepit) {    // remove it
    abag->setRemove();
}
else
    wcount++;
}

// Wait till the end to remove stuff to make sure all the variant stuff is
// taken care of
witer = htActivityBags->end();
for (htiter = htActivityBags->begin(); htiter != htActivityBags->end();
    htiter++) {
    if (witer != htActivityBags->end()) {
        htActivityBags->erase(witer);
        witer = htActivityBags->end();
    }
    abag = htiter->second;
    if (abag->getRemove()) witer = htiter;
}
if (witer != htActivityBags->end()) htActivityBags->erase(witer);

return wcount;
}

```

4.3 Sequence Mining of Temporal Clusters (SMTc)

Header.

```
#pragma once
#include <unordered_map>
#include "Parameters.h"
#include "Terms.h"
#include "Generics.h"
#include "Timeline.h"
#include "TimelineEntry.h"
#include "Strings.h"
#include "Patterns.h"
#include "TimelineEntry.h"
#include "ActivityBag.h"
#include "queues.h"
#include "utilities.h"
#define htActivityBag_i std::unordered_map<uint32_t, ActivityBag *>::const_iterator

class smtc
{
    Utilities *utilities;
    Parameters *parameters;
    Generics *generics;
    Patterns *patterns;
    Strings *strings;
    Terms *terms;
    Timeline *timeline;

    std::unordered_map<uint32_t, ActivityBag *> *htActivityBags;

    typedef struct treeKey_type {
        uint32_t timelineid;
        int termid;
        TimelineEntry *entry;
    } treeKey;

    typedef struct orderSortType {
        uint32_t position;
        uint32_t order;
    } orderSortList;

    typedef struct powerData_type {
        queueT<TimelineEntry> *head;
```

```

        uint32_t *orderList;
        orderSortList *ordered;
        uint32_t length;
        uint32_t maxLength;
        uint32_t incrLength;
        char *include;
    } powerData;

static bool tlEntryLessThan(const TimelineEntry *lhs, const TimelineEntry *rhs) {
    if (lhs->term < rhs->term) {return true; }
    if (lhs->term > rhs->term) {return false; }

    // terms equal
    if (lhs->timelineid < rhs->timelineid) {return true;}
    if (lhs->timelineid > rhs->timelineid) {return false; }

    // false if elements are equal
    return false;
}

public:
    smtc(Utillities *wutilities, Parameters *wparameters, Patterns *wpatterns,
          Strings *wstrings, Generics *wgenerics, Terms *wterms);
    ~smtc();
    void process();
};

C++.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Sequential Mining of Temporal Clusters (SMTC)                                         //
// Input: a dataset used to generate a set of production rules (terms&strings)//
// Output: a set of production rules                                                            //
// Notes:                                                                                         //
// Initial Release: 13-Jul-17                                                                    //
//                                                                                               //
// Revision History:                                                                              //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "smtc.h"

smtc::smtc(Utillities *wutilities, Parameters *wparameters, Patterns *wpatterns,
            Strings *wstrings, Generics *wgenerics, Terms *wterms) {

```

```

    printf("Initializing SMTC\n");
    utilities = wutilities;
    parameters = wparameters;
    patterns = wpatterns;
    strings = wstrings;
    terms = wterms;
    generics = wgenerics;
    htActivityBags = new std::unordered_map<uint32_t, ActivityBag *>;
}

smtc::~smtc() {}

void smtc::process() {
    queueT<TimelineEntry> *qhead;          // the workspace of events
    listT<TimelineEntry> *treeTimelineOrderList; // the workspace sorted by term,
                                              // timelineid
    queueT<TimelineEntry> *newHead;        // the workspace passed to
                                              // addActivityBag
    TimelineEntry *entry;                  // The current event (added at the end
                                              // of the processing)
    TimelineEntry *entry2;                 // used to work with the events in the
                                              // workspace
    orderSortList *worderSortList, tmpSortList; // sorted sequence of entries in
                                              // newHead
    uint32_t *worderList;                  // sorted sequence of entries in newHead
    uint32_t activeBags;                   // # of active trms passed to addActivityBag

    uint8_t isFirst;                       // First Pass
    uint32_t curEndTime;                   // End time of first entry in the workspace

    ActivityBag abag;                      // used just to call addActivityBag
                                              // (nothing's stored in it)
    ActivityBag *tmpAbag;                  // used to prune and sort activity bags (and
                                              // make strings from them)

    uint8_t isKeeper;                      // true if this activity bag meets the
                                              // threshold of "interesting"
    TermList *wterm, *prevTerm, *wtermHead; // used to create the term sequence
                                              // we want (i.e. the most frequent)

    // Used to eliminate loops in the workspace
    uint32_t oldTerm;
    uint32_t loopsFound;

```

```

// Used to figure out which sequence is the most frequent
int *stringOrder;
uint32_t *sourceList;
uint32_t maxCount, maxIndex, windex;

// local variables to speed processing (i.e., I don't have to keep calling
// abag to get these pointers)
int *wactivityTable;
uint32_t *wcountTable, *wsourceTable;

// Iterators and Counters
htActivityBag_i htiter;
void *qiter;
void *iterL;
uint32_t ctr, last_seen;
uint32_t i, j;
int k;
uint32_t totCtr;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Initialize Variables
treeTimelineOrderList = new listT<TimelineEntry>;
qhead = new queueT<TimelineEntry>;
newHead = new queueT<TimelineEntry>;

// Initialize processing of events -- get the first case number
timeline = new Timeline(utilities, parameters, terms, generics);
if (timeline->initialize() == NULL) return;

worderSortList = (orderSortList *)malloc(sizeof(orderSortList) *
                                           (parameters->maxString + 1));
worderList = (uint32_t *)malloc(sizeof(uint32_t) * (parameters->maxString + 1));

do {

    // Initialize the variables for this case
    isFirst = 1;
    ctr = 0;

```



```

last_seen = 0;
qhead->qclear();
treeTimelineOrderList->lclear();

// Iterate through all events for this case
while ((entry = timeline->process(1)) != NULL) {
    // Status Update
    if (utilities->showTime(0, &ctr, &last_seen, parameters->increment) )
        printf("\t\ttabagCount = %u\t\ttermCount = %u\t\ttreeCount = %u\n",
                parameters->abagCount, parameters->termCount,
                treeTimelineOrderList->getCount());

    // First time through -- get things started
    if (isFirst) {
        isFirst = 0;
        qhead->qpush(entry);
        curEndTime = entry->getEndTime();
        treeTimelineOrderList->insertSort(entry, &(smtc::tlEntryLessThan));
        continue;
    }

    // If we're still within the buffer, keep getting records
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "checking if still in scope\n");
    if (qhead->qcount() < parameters->maxString &&
        (entry->getStartTime() - curEndTime < parameters->bufferTime)) {
        qhead->qpush(entry);
        treeTimelineOrderList->insertSort(entry, &(smtc::tlEntryLessThan));
        continue;
    }

    // Otherwise, it's time to add abags
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "***time to add stuff***\n");

    // Iterate through the workspace, removing sequential terms that are
    // the same
    qiter = qhead->initializeIterator();
    oldTerm = 0;
    loopsFound = 0;
    while (qiter != NULL) {
        entry2 = qhead->qvalue(qiter);
        if (oldTerm != 0) {

```

```

        if (oldTerm == entry2->getTerm()) {
            loopsFound++;
            qiter = qhead->iterate(qiter);
            if (!(treeTimelineOrderList->lerase(entry2)))
                printf("couldn't remove wtkey1\n");
            if (!(qhead->qerase(entry2)))
                printf("couldn't remove wtkey1a\n");
            delete entry2;
            continue;
        }
    }
    oldTerm = entry2->getTerm();
    qiter = qhead->iterate(qiter);
}

// make sure the buffer is smaller than maximum size --
//      this will happen when we're still in time range, but the buffer
//      is full we do this stuff and then add another record.
while (!qhead->qempty() && qhead->qcount() > parameters->maxString) {
    entry2 = qhead->qpop();
    if (!(treeTimelineOrderList->lerase(entry2)))
        printf("couldn't remove wtkey2\n");
    delete entry2;
}

// Create an order by timelineid
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "Creating ordering\n");
i = 0;
qiter = qhead->initializeIterator();
while (qiter != NULL) {
    entry2 = qhead->qvalue(qiter);
    entry2->setOrder(i++);
    qiter = qhead->iterate(qiter);
}

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
            "Ready to start processing combinations\n");

// If we didn't eliminate the whole workspace when we eliminated loops
if (qhead->qcount() > 0) {
    unsigned int witerator; // this means maxString can't be more than 32

```

```

if (parameters->maxString > 32) {
    printf("**error** SMTC is restricted to strings <= 32 terms\n");
    return;
}

// Keep track of the number of events in the workspace to be
// included in the test string
activeBags = qhead->qcount();

// Go through every possible combination of the events in the
// workspace (i.e., the power set of the workspace)
// using iterator lets us quickly figure out which events are "on"
for (witerator = 2 ^ (qhead->qcount() - 1); witerator > 0; witerator--) {

    // Load the active entries into the queue
    i = 0;
    activeBags = 0;
    newHead->qclear();
    for (iterL = treeTimelineOrderList->getRear(); iterL != NULL;
        iterL = treeTimelineOrderList->iterate(iterL)) {

        // boundary check
        if (i >= qhead->qcount() ) break;

        // if the bit in iterator corresponding to the current event
        // is on, add it to the queue being passed
        if ( (((unsigned int)pow(2, i)) & witerator) != 0) {

            // add the term to the list
            entry2 = treeTimelineOrderList->lvalue(iterL);
            newHead->qpush(entry2);

            // Insert the new term in the right spot in the sorted
            // order list
            worderSortList[activeBags].order = entry2->getOrder();
            worderSortList[activeBags].position = activeBags;
            tmpSortList = worderSortList[activeBags];
            for (k = activeBags - 1;
                k >= 0 && worderSortList[k].order > tmpSortList.order;
                k--) {
                worderSortList[k + 1] = worderSortList[k];
            }
            worderSortList[k + 1] = tmpSortList;

```

```

        // Increment the number of active abags
        activeBags++;
    }
    i++;
}

// Create the order list from the sorted term list
for (i = 0; i < activeBags; i++)
    worderList[i] = worderSortList[i].position;

// Add the activity bag
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "Entering addAbag -- activeBags = %i\n", activeBags);
abag.addActivityBag(newHead, worderList, activeBags, 0, 1,
    strings, htActivityBags, parameters, utilities, patterns);
}

}

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "Finished processing combinations\n");

// Get rid of all of the records in the buffer that are outside the new
// buffer
while (!qhead->qempty()) {
    entry2 = qhead->qvalue(qhead->initializeIterator());
    if (entry->getStartTime() - entry2->getEndTime() >=
        parameters->bufferTime) {
        if (!(treeTimelineOrderList->lerase(entry2)))
            printf("couldn't remove wtkey3\n");
        qhead->qpop();
        delete entry2;
    }
    else
        break;
}

// Connect the new record to the linked list
qhead->qpush(entry);
treeTimelineOrderList->insertSort(entry, &tlEntryLessThan);
}

```

```

// Remove everything remaining in the queue
while (!qhead->qempty()) {
    entry2 = qhead->qpop();
    if (!(treeTimelineOrderList->lerase(entry2)))
        printf("couldn't remove wtkey3\n");
    delete entry2;
}

// Get the next record
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "getting the next record\n");
if (parameters->debugParms >= 5 && entry != NULL)
    fprintf(parameters->logFile, "term = %i    activity = %i\n",
        (int)entry->getTerm(), (int)entry->getActivity());
} while (timeline->iterate());

if (1) printf("\n\n\n***pruning the activities***\n");
ctr = totCtr = last_seen = 0;

// Iterate through each activity bag
//    -- although there is an "order" of terms, it's an arbitrary order
for (htiter = htActivityBags->begin(); htiter != htActivityBags->end();
    htiter++) {
    tmpAbag = htiter->second;

    // Show progress
    if (utilities->showTime(0, &totCtr, &last_seen, parameters->increment))
        printf("\t\tRemoved = %u\n", ctr);

    // Determine if the abag meets the threshold to be kept
    isKeeper = (((float)(tmpAbag->getCount() * (pow(tmpAbag->getTotalLength()
        / tmpAbag->getGenericCount(), parameters->epsilon)))
        < (float)parameters->thresholdSMTC) || (tmpAbag->getCount() == 1) ||
        (tmpAbag->getLength() == 1) ? 0 : 1);

    // If they want to see the scores, show it to them
    if (parameters->showValues) {
        wterm = tmpAbag->getActivities();
        fprintf(parameters->logFile, "activities:  ");
        while (wterm != NULL) {

```

```

        fprintf(parameters->logFile,"%i
            ", wterm->getTerm());    wterm = wterm->getNext();
    }
    fprintf(parameters->logFile, "\n");
    fprintf(parameters->logFile,
        "Threshold = %f    count = %u    length = %u    genCount = %u    value = %f
        (float)parameters->thresholdSMTC, tmpAbag->getCount(),
        tmpAbag->getTotalLength(), tmpAbag->getGenericCount(),
        (float)(tmpAbag->getCount() * (pow(tmpAbag->getTotalLength() /
        tmpAbag->getGenericCount(), parameters->epsilon)))));
    }

    // If it's not a keeper remove it
    if (!isKeeper) {
        ctr++;
        tmpAbag->setRemove();
    }
}
if (1) printf("\t%u of %u activity bags removed\n", ctr, totCtr);

//   for each remaining abag:
//       sort the activities somehow (k-means and levenshtein) and
//       create DB strings for them (possibly multiple orderings with single
//       string id)
if (1) printf("\n\n***Sorting remaining activity bags***\n");
ctr = totCtr = last_seen = 0;
for (htiter = htActivityBags->begin(); htiter != htActivityBags->end();
    htiter++) {
    tmpAbag = htiter->second;

    // boundary check
    if ((tmpAbag->getLength() <= 0) || (tmpAbag->getRemove() == 1)) continue;

    // show progress
    utilities->showTime(1, &ctr, &last_seen, parameters->increment);

    // Initialize variables
    stringOrder = (int *)malloc(sizeof(int) * tmpAbag->getLength());
    if (stringOrder == NULL) {
        printf("***smtc -- unable to allocated stringOrdered -- returning\n");
        return;
    }
}

```

```

sourceList = (uint32_t *)malloc(sizeof(uint32_t) * tmpAbag->getLength());
if (sourceList == NULL) {
    printf("**smtc -- unable to allocated sourceList -- returning\n");
    return;
}
wcountTable = tmpAbag->getCountTable();
wactivityTable = tmpAbag->getActivityTable();
wsourceTable = tmpAbag->getSourceTable();
if (wsourceTable == NULL || wactivityTable == NULL || wcountTable == NULL)
    continue;

// Figure out the ordering that occurs the most (within tree edit distance).
// While that ordering meets the above threshold, create a stringid and
// string records for the ordering and its tree edit distance variations
for (i = 0; i < tmpAbag->getLength(); i++) {
    maxCount = 0;
    stringOrder[i] = 0;
    sourceList[i] = 0;
    maxIndex = 0;
    for (j = 0; j < tmpAbag->getLength(); j++) {
        windex = j * tmpAbag->getLength() + i; // # times activity j in pos i
        if (windex < tmpAbag->getLength() * tmpAbag->getLength()) {
            if (wcountTable[windex] > maxCount) {
                stringOrder[i] = wactivityTable[j]; // activity most in pos i
                sourceList[i] = wsourceTable[j];
                maxCount = wcountTable[windex];
                maxIndex = j;
            }
        }
    }
    for (j = 0; j < tmpAbag->getLength(); j++)
        wcountTable[(maxIndex * tmpAbag->getLength()) + j] = 0;
}

// Change the term list for the bag to the one we want
prevTerm = NULL;
wtermHead = NULL;
for (i = 0; i < tmpAbag->getLength(); i++) {
    wterm = new TermList(stringOrder[i], sourceList[i], i, NULL, NULL);
    if (prevTerm != NULL) prevTerm->setNext(wterm);
    if (wtermHead == NULL) wtermHead = wterm;
    prevTerm = wterm;
}

```

```
        wterm = tmpAbag->getActivities();
        delete wterm;
        tmpAbag->setActivities(wtermHead);
    }

    // At the end, make sure the DB files are updated
    abag.createString(htActivityBags,strings);
    generics->writeGenericItems();
    patterns->writePatterns();
}
```


4.4 String Abstraction (AbstractStrings)

Header.

```
#pragma once
#include "TearCPP.h"
#include "Parameters.h"
#include "Patterns.h"
#include "Strings.h"
#include "TermList.h"
#include "Generics.h"
#include "TimelineEntry.h"
#include "queues.h"
#include "stacks.h"
#include "lists.h"
#include "utilities.h"

class AbstractStrings
{
    Utilities *utilities;
    Parameters *parameters;
    Patterns *patterns;
    Strings *strings;
    Generics *generics;
    Terms *terms;

    // Counters for the next timeline and string instance
    uint32_t maxtimelineid;
    uint32_t nextInstance;

public:
    AbstractStrings(Utilities *wutilities, Parameters *parameters,
                   Patterns *patterns, Strings *strings,
                   Generics *generics, Terms *wterms);
    ~AbstractStrings();
    void AbstractStrings::process();
};

C++.
```

```
////////////////////////////////////
//                                                                    //
// Class: AbstractStrings                                           //
// Input: dataset to be abstracted using the production rules (strings) in db //
```

```

// Output: a dataset abstracted using the production rules //
// Notes: 1. The timelineid of the output dataset is 5 more than the //
//         timelineid of the inputted dataset. //
//         2. Dump files are used because DB updates within the program slow it//
//         to a crawl. This means that after the program completes the dump //
//         files need to be imported into the DB and then deleted. //
// Initial Release: 13-Jul-17 //
// //
// Revision History: //
// //
////////////////////////////////////
#include "AbstractStrings.h"

AbstractStrings::AbstractStrings(Utilities *wutilities, Parameters *wparameters,
                                Patterns *wpatterns, Strings *wstrings,
                                Generics *wgenerics, Terms *wterms) {
    char exec_string[2000];
    int rc;
    sqlite3_stmt *stmt;

    utilities = wutilities;
    parameters = wparameters;
    patterns = wpatterns;
    strings = wstrings;
    generics = wgenerics;
    terms = wterms;

    // Get the next timelineid to add
    sprintf_s(exec_string, 2000, "SELECT max(timelineid) FROM timeline;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    rc = sqlite3_step(stmt);
    if (rc != SQLITE_ROW) return;
    maxtimelineid = sqlite3_column_int(stmt, 0) + 1;
    sqlite3_finalize(stmt);

    // Get the next string instance to add
    sprintf_s(exec_string, 2000, "SELECT MAX(instanceid) FROM stringInstances;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    rc = sqlite3_step(stmt);
    nextInstance = sqlite3_column_int(stmt, 0) + 1;
    sqlite3_finalize(stmt);
}

```

```

AbstractStrings::~~AbstractStrings() {} // nothing special to cleanup

void AbstractStrings::process() {

    // file handles for the records to be dumped
    FILE *timelinefile, *stringInstances, *stringSupport,
        *stringXref, *stringXrefSupport;

    Timeline *timeline;                // Timeline class used to get each event
    TimelineEntry *entry;              // The current event being examined
    TimelineEntry *entry2;
    TimelineEntry *newEntry;           // The new event added (occurs before the
                                        // current event is put in the workspace)

    queueT<TimelineEntry> *qhead;       // The workspace of events current being
                                        // examined

    // Used to create a list of items linked to terms that are sent to calcString
    queueT<ItemInstance> *qItems;       // A queue of the items in the workspace
    stackT<ItemInstance> *sitems;       // A stack of the items in the workspace
                                        // (makes checking smaller and smaller
                                        // strings easier)

    ItemInstance *witem;

    // Used to create a list of terms to be check to see if a string
    TermList *wtermList;               // The termlist being check to see if it's a
                                        // string
    stackT<TermList> *sterms;          // A stack of the terms in the workspace --
                                        // allows peeling off one term at a time
    TermList *wterm, *prevTerm;        // Temporary variables Used to create the
                                        // termlist
    TermList *newTerms;                // The results from calcString (if non-null,
                                        // it forms a string)
    uint32_t sid;                      // The stringid for these newTerms

    // Cross reference lists for generic items created when string is calculated
    listT<GenericXref> *wgenXref;       // Generic item to actual item

    // Generic item for new string to generic item of old strings in new string
    listT<InstanceGenericXref> *headInstanceXref;
    GenericXref *tmpGenXref;

```

```

InstanceGenericXref *tmpInstanceGenXref;

char *scaseNbr;           // The current case
uint8_t isFirst;          // True if this is 1st record being processed
bool skipit;              // True if the currently sized workspace can't
                           // be converted to a string
uint32_t curEndTime;      // Earliest endtime of an event in workspace
uint32_t wsize;           // Size of the subsection of the workspace
                           // being examined for a string
uint32_t lowTime, highTime; // Earliest and Latest times for the events
                           // that make up the new string instance

// There can only be one non-zero actor in a string. These variables check
uint32_t actorInstance;
uint32_t wactor;

// Iterators and counters
uint32_t i, j;
void *qiter, *siter, *liter;

////////////////////////////////////

// Initialize variables
timeline = new Timeline(utilities, parameters, terms, generics);
scaseNbr = timeline->initialize();
if (scaseNbr == NULL) return;

qhead = new queueT<TimelineEntry>;
qlItems = new queueT<ItemInstance>;
sterms = new stackT<TermList>;
sitems = new stackT<ItemInstance>;

do {
    // For each case we're opening and closing the output files to ensure
    // everything is written and safe
    fopen_s(&timelinefile, ".\\timeline.out", "a");
    fopen_s(&stringInstances, ".\\stringInstances.out", "a");
    fopen_s(&stringSupport, ".\\stringSupport.out", "a");
    fopen_s(&stringXref, ".\\stringXref.out", "a");
    fopen_s(&stringXrefSupport, ".\\stringXrefSupport.out", "a");

```

```

// Initialize variables for this new case
printf("Processing %s\n", scaseNbr);
isFirst = 1;
qhead->qclear();

// For each event in the current case
while ((entry = timeline->process(0)) != NULL) {
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "**AbstractStrings -- activity %i -- actor = %i    action = %i    app = %i\n",
            entry->getActivity(), entry->getActor(), entry->getAction(),
            entry->getAppexec(), entry->getItemtype(),
            entry->getItems()->getItem(), entry->getTerm());

    // Start us off
    if (isFirst) {
        isFirst = 0;
        qhead->qpush(entry);
        curEndTime = entry->getEndTime();
        continue;
    }

    // If we're still within the buffer, keep getting records
    if (qhead->qcount() < parameters->maxString && entry->getStartTime() -
        curEndTime < parameters->bufferTime) {
        qhead->qpush(entry);
        continue;
    }

    // Otherwise,
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "***time to add stuff***    wlength = %i\n", qhead->qcount());

    // Set up the term list and item list
    prevTerm = NULL;
    wtermList = NULL;
    qiter = qhead->initializeIterator();
    sterm->sclear();
    sitem->sclear();
    qitem->qclear();

    // Create a stack of terms and items so we can pull them off

```

```

// LIFO to loop through smaller and smaller candidates
// till we find a match
i = 0;
for (qiter = qhead->initializeIterator(); qiter != NULL;
     qiter = qhead->iterate(qiter)) {
    entry2 = qhead->qvalue(qiter);

    // within the term stack, the records are linked
    // FIFO for CalcString processing
    wterm = entry2->getTermList();
    wterm->setNext(NULL);
    wterm->setOrder(i++);
    if (prevTerm != NULL) prevTerm->setNext(wterm);
    if (wtermList == NULL) wtermList = wterm;
    prevTerm = wterm;
    sterms->spush(wterm);

    witem = entry2->getItems();
    qItems->qpush(witem);
    sitems->spush(witem);
}

// Now try to create strings from longest to shortest
// (worse case, create a single element activity)
newEntry = NULL;
for (wsize = qhead->qcount(); wsize >= 1; wsize--) {
    if (wsize == 1) { // no string, just copy the
        // timeline record. Terms don't exist for
        // everything, e.g., activities that only exist a
        // few times -- in that case, just leave
        // the event as is; it doesn't get abstracted
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "adding individual event\n");

        // Remove the last term and entry from the queues
        wterm = sterms->spop();
        entry2 = qhead->qpop();

        // Copy this single event into new dataset
        fprintf(timelinefile,
            "%u|%u|%u|%u|%u|i|i|u|i|i|u|s\n",
            maxtimelineid++, // Timelineid

```

```

        parameters->existingLevel + 5,    // Timelinetypeid
        entry2->getActivity(),           // Activity1
        0,                               // Activity2
        entry2->getRelationship(),        // Relationship
        entry2->getStartTime(),           // Start Time
        entry2->getEndTime(),             // End Time
        entry2->getSourceID(),            // Sourceid
        entry2->getTraceType(),           // Tracetype
        entry2->getTraceid(),             // TraceID
        scaseNbr != NULL ? scaseNbr : " "); // Case Number
delete entry2;
break;

} else { // more than one record in the workspace

    // Ensure all generic actors are the same
    skipit = false;
    wactor = 0;
    for (qiter = qhead->initializeIterator();
         qiter != NULL; qiter = qhead->iterate(qiter)) {
        actorInstance = qhead->qvalue(qiter)->getActor();

        // right now it looks like N/A is the only
        // exception to 1 actor per string
        if (actorInstance == 0) continue;
        if (wactor == 0) wactor = actorInstance;
        else if (actorInstance != wactor) {
            skipit = true;
            break;
        }
    }
    if (skipit) {
        // shorten the term and item lists by one
        wterm = sterms->spop();
        siter = sterms->getTop();
        wterm = sterms->svalue(siter);
        wterm->setNext(NULL);
        qItems->qpopRear();
        continue;
    }

    // check if all the remaining terms in the queue make a string
    wgenXref = new listT<GenericXref>;

```

```

headInstanceXref = new listT<InstanceGenericXref>;
newTerms = strings->calcString(wtermList, qItems, wsize, 0,
                               wgenXref, headInstanceXref, patterns);

// shorten the term and item lists by one
wterm = sterms->spop();
siter = sterms->getTop();
wterm = sterms->svalue(siter);
wterm->setNext(NULL);
qItems->qpopRear();

// If we didn't get any potential string, remove the cross
// reference stuff and go to the next smaller size
if (newTerms == NULL) {
    for (liter = wgenXref->getFront(); liter != NULL;
         liter = wgenXref->revIterate(liter)) {
        tmpGenXref = wgenXref->lvalue(liter);
        if (tmpGenXref != NULL) delete tmpGenXref;
    }
    delete wgenXref;

    for (liter = headInstanceXref->getFront(); liter != NULL;
         liter = headInstanceXref->revIterate(liter)) {
        tmpInstanceGenXref = headInstanceXref->lvalue(liter);
        if (tmpInstanceGenXref != NULL)
            delete tmpInstanceGenXref;
    }
    delete headInstanceXref;

    continue;
}

// Get the string id
sid = strings->getString(newTerms);

if (sid > 0) {
    while (!(sterms->empty())) {
        wterm = sterms->spop();
    }

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "adding support\n");
}

```



```

nextInstance++;

// Add String Instance
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\tadding string instance %u for string %u\n",
        nextInstance, sid);
fprintf(stringInstances,
    "%u|%u\n", nextInstance, sid);

// Add String Xrefs
lowTime = MAXINT;
highTime = 0;
actorInstance = 0;
for (j = 0; j < wsize; j++) {
    entry2 = qhead->qpop();

    // all the non-zeros are the same
    if (actorInstance == 0)
        actorInstance = entry2->getActor();
    if (entry2->getStartTime() < lowTime)
        lowTime = entry2->getStartTime();
    if (entry2->getEndTime() > highTime)
        highTime = entry2->getEndTime();

    fprintf(stringXref, "%u|%u|%u\n", nextInstance, j,
        entry2->getTimelineID());

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "\tadding string xref for timeline %u (order %u) to instance %u\n",
            entry2->getTimelineID(), j, nextInstance);
    delete entry2;
}

// Create a string support record for the actor
fprintf(stringSupport,
    "%u|%u|%u|%u\n", nextInstance, GNRC_ACTOR,
    GNRC_ACTOR_INSTANCE, actorInstance);

```

```

// Add String Support
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "wsize = %u\n", wsize);
for (liter = wgenXref->getFront(); liter != NULL;
    liter = wgenXref->revIterate(liter)) {
    tmpGenXref = wgenXref->lvalue(liter);
    fprintf(stringSupport, "%u|%u|%u|%u\n",
        nextInstance, tmpGenXref->getGenericType(),
        tmpGenXref->getGeneric(),
        tmpGenXref->getGenericType() == GNRC_ITEM ?
            tmpGenXref->getItem() :
            tmpGenXref->getGenericType() ==
                GNRC_ITEMLIST ? tmpGenXref->getListid() : 0);
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "\tadding string support with instance %u, generic %u i
                nextInstance, tmpGenXref->getItem(),
                tmpGenXref->getGenericType() == GNRC_ITEM ?
                tmpGenXref->getItem() :
                tmpGenXref->getGenericType() == GNRC_ITEMLIST ?
                tmpGenXref->getListid() : 0);
    delete tmpGenXref;
}
delete wgenXref;

// Add Generic String Xref Support
for (liter = headInstanceXref->getFront(); liter != NULL;
    liter = headInstanceXref->revIterate(liter)) {
    tmpInstanceGenXref = headInstanceXref->lvalue(liter);
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "order %s    old: %u    new: %u\n",
            tmpInstanceGenXref->getOrder(),
            tmpInstanceGenXref->getOldGeneric(),
            tmpInstanceGenXref->getNewGeneric());

    fprintf(stringXrefSupport, "%u|%s|%u|%u\n",
        nextInstance, tmpInstanceGenXref->getOrder(),
        tmpInstanceGenXref->getOldGeneric(),
        tmpInstanceGenXref->getNewGeneric());
    delete tmpInstanceGenXref;
}
delete headInstanceXref;

```

```

delete newTerms;

// Add new timeline record
fprintf(timelinefile,
    "%u|%u|%u|%u|%u|i|i|i|i|i|s\n",
    maxtimelineid++,           // Timelineid
    parameters->existingLevel + 5, // Timelinetypeid
    nextInstance,             // Activity1
    0,                        // Activity2
    R_BETWEEN,                // Relationship
    lowTime,                  // Start Time
    highTime,                 // End Time
    S_STRING,                 // Sourceid
    nextInstance,             // Tracetype
    sid,                      // TraceID
    scaseNbr != NULL ? scaseNbr : " "); // Case Number

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\tadding timeline record for instance %u\n",
        nextInstance);
break;

} else {

delete newTerms;

for (liter = wgenXref->getFront(); liter != NULL;
    liter = wgenXref->revIterate(liter)) {
    tmpGenXref = wgenXref->lvalue(liter);
    if (tmpGenXref != NULL) delete tmpGenXref;
}
delete wgenXref;

for (liter = headInstanceXref->getFront(); liter != NULL;
    liter = headInstanceXref->revIterate(liter)) {
    tmpInstanceGenXref = headInstanceXref->lvalue(liter);
    if (tmpInstanceGenXref != NULL) delete tmpInstanceGenXref;
}
delete headInstanceXref;

} // if sid != -1 else

```

```

        } // if qhead->qcount == 1 else
    } // for

    if (!qhead->qempty()) {
        qiter = qhead->initializeIterator();
        curEndTime = (qhead->qvalue(qiter))->getEndTime();
    } else
        curEndTime = entry->getEndTime();

    qhead->qpush(entry);

}

while ( (entry = qhead->qpop()) != NULL) {
    fprintf(timelinefile, "%u|%u|%u|%u|%u|%i|%i|%u|%i|%u|%s\n",
            maxtimelineid++,           // Timelineid
            parameters->existingLevel + 5, // Timelinetypeid
            entry->getActivity(),        // Activity1
            0,                          // Activity2
            entry->getRelationship(),    // Relationship
            entry->getStartTime(),       // Start Time
            entry->getEndTime(),         // End Time
            entry->getSourceID(),        // Sourceid
            entry->getTraceType(),       // Tracetype
            entry->getTraceid(),         // TraceID
            scaseNbr != NULL ? scaseNbr : " "); // Case Number
}

fclose(timelinefile);
fclose(stringInstances);
fclose(stringSupport);
fclose(stringXref);
fclose(stringXrefSupport);

} while (scaseNbr = timeline->iterate());

// write the generics and patterns to the database
generics->writeGenericItems();
patterns->writePatterns();
}

```

4.5 Loop Abstraction (AbstractLoops)

Header.

```
#pragma once
#include "TearCPP.h"
#include "Parameters.h"
#include "Strings.h"
#include "TermList.h"
#include "Generics.h"
#include "TimelineEntry.h"
#include "queues.h"
#include "Utilities.h"

class AbstractLoops
{
    Utilities *utilities;
    Parameters *parameters;
    Strings *strings;
    Generics *generics;
    Terms *terms;

    // The next timelineid and loopid
    uint32_t maxtimelineid;
    uint32_t nextLoop;

private:
    void CreateLoop(queueT<TimelineEntry> *head, uint32_t lowTime,
                    uint32_t highTime, FILE *timelinefile, FILE *loopfile,
                    FILE *loopSupport);

public:
    AbstractLoops(Utilities *wutilities, Parameters *wparameters,
                  Strings *wstrings, Generics *wgenerics, Terms *wterms);
    ~AbstractLoops();
    void process();
};
```

C++.

```
////////////////////////////////////
//                                                                    //
// Class: AbstractLoops                                              //
// Input: a dataset that has been abstracted using production rules (strings) //
```

```

// Output: a dataset where consecutive entries with same operator have been //
//         collapsed into loops. //
// Notes: 1. Timelinetypeid of output dataset is 5 more than timelinetypeid //
//         of the inputted dataset. //
//         2. Dump files are used because DB updates within the program slow it//
//         to a crawl. This means that after the program completes the dump //
//         files need to be imported into the DB and then deleted. //
// Initial Release: 13-Jul-17 //
// //
// Revision History: //
// //
////////////////////////////////////
#include "AbstractLoops.h"

AbstractLoops::AbstractLoops(Utilities *wutilities, Parameters *wparameters,
                             Strings *wstrings, Generics *wgenerics, Terms *wterms) {
    char exec_string[2000];
    int rc;
    sqlite3_stmt *stmt;

    utilities = wutilities;
    parameters = wparameters;
    strings = wstrings;
    generics = wgenerics;
    terms = wterms;

    // Get the next timelineid
    sprintf_s(exec_string, 2000, "SELECT max(timelineid) FROM timeline;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    rc = sqlite3_step(stmt);
    if (rc != SQLITE_ROW) return;
    maxtimelineid = sqlite3_column_int(stmt, 0);
    sqlite3_finalize(stmt);

    // Get the next loopid
    sprintf_s(exec_string, 2000, "SELECT MAX(loopid) FROM loops;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    rc = sqlite3_step(stmt);
    nextLoop = sqlite3_column_int(stmt, 0) + 1;
    sqlite3_finalize(stmt);
}

```

```

AbstractLoops::~AbstractLoops() {} // Nothing special to do

void AbstractLoops::process() {

    FILE *timelinefile, *loopfile, *loopSupport;    // Output files
    Timeline *timeline;                            // Class for retrieving the timeline entries
    TimelineEntry *entry;                          // The current timeline entry being examined
    TimelineEntry *newEntry;                       // The entry to be added to the next level
                                                    // (added before entry is put in the queue)
    queueT<TimelineEntry> *head;                   // The workspace of entries currently being
                                                    // checked for a loop

    char *scaseNbr;                                // The current case being examined

    uint32_t curEndTime;                           // Earliest end time for records in workspace
    bool outOfTime;                                // true if the elapsed time between curEndTime
                                                    // and the new record is too long
    uint32_t lowTime, highTime;                    // first and last timestamp for current loop

    // Values for the current event
    uint32_t wsource, wactor, waction, wappexec, witemtype, stime, etime;
    int wtermid;

    // Values for the first record in the workspace
    uint32_t oldSource, oldActor, oldAction, oldAppexec, oldItemtype;
    int oldtermid;

    bool processIt = false;
    ///////////////////////////////////////////////////////////////////

    // Initialize the timeline for inputting the current dataset
    timeline = new Timeline(utilities, parameters, terms, generics);
    scaseNbr = timeline->initialize();
    if (scaseNbr == NULL) return;

    // Initialize the queues
    head = new queueT<TimelineEntry>;

    // Initialize the variables
    curEndTime = oldActor = oldAction = oldAppexec = oldItemtype = oldSource =
        oldtermid = 0;
    lowTime = 0; highTime = MAXINT;

```

```

do { // Do for each case
    fopen_s(&timelinefile, ".\\timeline.out", "a");
    fopen_s(&loopfile, ".\\loops.out", "a");
    fopen_s(&loopSupport, ".\\loopSupport.out", "a");

    // for each entry in the current case
    while ((entry = timeline->process(0)) != NULL) {

        // Get the current values (slightly faster than multiple calls to the
        // get functions)
        wtermid = entry->getTerm();
        wsource = entry->getSourceID();
        wactor = entry->getActor();
        waction = entry->getAction();
        wappexec = entry->getAppexec();
        witetype = entry->getItemtype();
        stime = entry->getStartTime();
        etime = entry->getEndTime();

        // Check if the elapsed time between the current event and the first
        // event in the workspace is too long
        outOfTime = ((lowTime != 0 && stime - curEndTime >
            parameters->bufferTime) ? true : false);

        // Check if its time to process the workspace
        // 1. current record is a loop (we don't do loops of loops)
        // 2. current record is a string and previous record isn't
        // 3. current record and previous record are strings, but the terms
        //    are different
        // 4. neither record is a string but the actor, action, application,
        //    or itemtype changed
        // 5. elapsed time from first event in workspace is too long
        if (wsource == S_LOOP) processIt = true;
        if (wsource == S_STRING && (oldSource != S_STRING ||
            wtermid != oldtermid || outOfTime || wactor != oldActor))
            processIt = true;
        if (wsource != S_LOOP && wsource != S_STRING && wsource != oldSource)
            processIt = true;
        if (wsource != S_LOOP && wsource != S_STRING && wtermid != 0 &&
            wtermid != oldtermid)
            processIt = true;
    }
}

```



```

if (wsource != S_LOOP && wsource != S_STRING &&
    (wactor != oldActor || waction != oldAction ||
     wappexec != oldAppexec || witemtype != oldItemtype))
    processIt = true;
if (outOfTime) processIt = true;

if (processIt) {
    // If there's more than one record in the workspace, create a loop
    // (and clear the workspace)
    // If there's exactly one, copy that event to the new level (and
    // remove it from the workspace)
    if (!head->qempty()) {
        if (head->qcount() > 1) {
            // create loop from head
            CreateLoop(head, lowTime, highTime, timelinefile, loopfile,
                       loopSupport);
        } else {
            newEntry = head->qpop(); // get off head queue before deleting

            fprintf(timelinefile, "%u|%u|%u|%u|%u|%i|%i|%u|%i|%u|%s\n",
                    ++maxtimelineid, parameters->existingLevel + 5,
                    newEntry->getActivity(), 0, newEntry->getRelationship(),
                    newEntry->getStartTime(), newEntry->getEndTime(),
                    newEntry->getSourceID(), newEntry->getTerm(),
                    newEntry->getTraceid(), newEntry->getCaseNbr());
            delete newEntry;
        }
    }

    // Add the current event to the workspace (it's only event there)
    head->qpush(entry);

    // Reset the values for the first record in the workspace
    lowTime = stime;
    highTime = etime;
    if (highTime < lowTime) highTime = lowTime;
    oldActor = wactor;
    oldAction = waction;
    oldAppexec = wappexec;
    oldItemtype = witemtype;
    oldtermid = wtermid;
    oldSource = wsource;
    curEndTime = etime;

```

```

    } else { // not yet time to process the workspace

        // Add the current event to the workspace
        head->qpush(entry);

        // If appropriate, update end times for the events in the workspace
        if (entry->getStartTime() < lowTime) lowTime = stime;
        if (entry->getEndTime() > highTime) highTime = etime;

        } // is it time to process the workspace?
    } // each event in the current case
    fclose(timelinefile);
    fclose(loopfile);
    fclose(loopSupport);

} while (scaseNbr = timeline->iterate()); // move to the next case

// write the generics to the database
generics->writeGenericItems();

}

void AbstractLoops::CreateLoop(queueT<TimelineEntry> *head, uint32_t lowTime,
                               uint32_t highTime, FILE *timelinefile, FILE *loopfile,
                               FILE *loopSupport) {
    uint32_t wloopid;           // the new loopid
    TimelineEntry *entry;       // the events to be put in the loop
    TimelineEntry *wtail;       // the event just added (and that can now be deleted)
    uint32_t firstTimeline;      // keep track of first timelineid for the loop record
    int wterm;                   // the loop's term

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "Creating Loop\n");
    if (head == NULL) return; // should never happen

    // Initialize variables
    firstTimeline = 0;
    if (lowTime > highTime) highTime = lowTime;
    wloopid = nextLoop++;

```

```

// Add loop support records
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\tAdding loop support\n");
wtail = NULL;
while ((entry = head->qpop()) != NULL) {
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "\t\tAdding timeline %i\n", entry->getTimelineID());
    if (wtail != NULL) delete wtail;
    if (firstTimeline == 0) firstTimeline = entry->getTimelineID();
    fprintf(loopSupport, "%u|%u\n", wloopid, entry->getTimelineID());
    wtail = entry;
}

// We hold onto the last event for things to put in the loop record
if (wtail != NULL && wtail->getTerm() != 0) wterm = wtail->getTerm();
else if (wtail != NULL) {
    wterm = terms->getGenericTermA(wtail->getActor(), wtail->getAction(),
        wtail->getAppexec(), wtail->getItemtype(), 0);
}
else // this shouldn't ever happen
    wterm = -1;

// Add a looprecord
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\tCreating loop record for term %i\n", wterm);
fprintf(loopfile, "%u|%u|%u|%u|i|%u|%u|%u\n",
    wloopid, firstTimeline, wtail->getTimelineID(),
    parameters->existingLevel + 5, wterm, lowTime, highTime,
    wtail->getSourceID());

// Create timeline record
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\tCreating timeline record for loop\n");
fprintf(timelinefile, "%u|%u|%u|%u|%u|i|i|%u|i|%u|s\n",
    ++maxtimelineid, parameters->existingLevel + 5,
    wloopid, 0, R_BETWEEN, lowTime, highTime, S_LOOP, wterm,
    wtail->getTraceid(), wtail->getCaseNbr());

// remove the last event from the workspace
if (wtail != NULL) delete wtail;
}

```

Appendix E. Scripts

5.1 Use Case Dataset

```
@echo off
echo the first input to this batch file is the location for files to be copied to
echo the second input to this batch file is the root drive of the mounted seized comp
echo mount the seized computer (e.g., through FTK Imager) and make sure the dates in
echo this script assumes the root directory of the seized computer is %2.  If not, ex
pause
```

```
rem Create the directories
```

```
mkdir %1\eventlogs
```

```
mkdir %1\registry
```

```
mkdir %1\prefetch
```

```
mkdir %1\user
```

```
rem Copy (most of) the files from the virtual machine
```

```
copy "%2\windows\system32\winevt\logs\*.*" %1\eventlogs
```

```
copy "%2\windows\system32\config\*.*" %1\registry
```

```
copy "%2\windows\prefetch\*.*" %1\prefetch
```

```
copy "%2\users\user\NTUSER.*" %1\user
```

```
copy "%2\users\user\appdata\local\microsoft\windows\UsrClass.*" %1\user
```

```
copy "%2\users\user\appdata\local\microsoft\windows\history\history.ie5\*.*"
%1\user
```

```
copy "%2\users\user\appdata\local\google\chrome\user data\default\History"
%1\chrome.sql
```

```
rem Create the master event messages database
```

```
rip -r %1\registry\system -p eventmsg >%1\filelist.txt
```

```
copy eventmessages.sql.orig eventmessages.sql
```

```
echo Global search and replace %windows%, %2\windows, etc.
```

```
pause
```

```
extractEventMessages %1\filelist.txt
```

```
copy eventmessages.sql %1\eventmessages.sql
```

```
rem Extract the information from the registry
```

```
rip -r %1\registry\software -p tearextract >%1\software_reg.txt
```

```
rip -r %1\registry\system -p tearextract >%1\system_reg.txt
```

```
rip -r %1\registry\sam -p tearextract >%1\sam_reg.txt
```

```
rip -r %1\registry\security -p tearextract >%1\security_reg.txt
```

```
rip -r %1\registry\default -p tearextract >%1\default_reg.txt
```

```
rip -r %1\registry\components -p tearextract >%1\components_reg.txt
```

```
rip -r %1\registry\software -p datetime >%1\software2_reg.txt
```

```

rip -r %1\registry\system -p datetime >%1\system2_reg.txt
rip -r %1\registry\sam -p datetime >%1\sam2_reg.txt
rip -r %1\registry\security -p datetime >%1\security2_reg.txt
rip -r %1\registry\default -p datetime >%1\default2_reg.txt
rip -r %1\registry\components -p datetime >%1\components2_reg.txt
rip -r %1\user\ntuser.dat -p tearextract >%1\ntuser_user_reg.txt
rip -r %1\user\usrclass.dat -p tearextract >%1\usrclass_user_reg.txt
rip -r %1\user\ntuser.dat -p datetime >%1\ntuser2_user_reg.txt
rip -r %1\user\usrclass.dat -p datetime >%1\usrclass2_user_reg.txt

rem Copy the master file table information from the virtual machine
mmls \\.\\physicaldrive1

echo
echo "fls -m g: -r -o XXXXXX -f ntfs \\.\\physicaldrive1 >%1\%1_mactime.txt"

rem Copy the firefox information from the virtual machine
echo copy places.sqlite from %2\users\user\appdata\roaming\mozilla\
echo          firefoxprofiles\xxxxx to c:firefox.sqlite

rem Extract the event log information from the (active) virtual machine
echo copy the files in eventlog and logparser to a shared drive with the
echo seized computer
echo start the seized computer and from the logparser directory:
echo          logparser "select timegenerated, eventid, sourcename, sid, message
echo          from *.evtx" -i:evt -o:tsv >%1\%1_eventlog.txt
echo then copy eventlog.txt and ensure the summary information has been
echo          removed from %1_eventlog.txt

```

5.2 Real World Dataset

```

#####
# Name: extractData.sh
# Author: James Okolica
# Date: 18-Aug-2016
#
# Purpose: Extract the digital forensic artifacts from the NTFS file system
#
# Process:
# 1. The target computer is shutdown and then started with an Ubuntu 14.04 install di
# 2. The analyst selects "Try Ubuntu (without installing) giving access to the system
# 3. The analyst mounts the remote file system
#          sudo mkdir /mnt/remote

```

```

#      sudo mount.cifs "//ss-lissard2-01/rdc(pii)/okolica/lissard"
#                               /mnt/remote -o user=jokolica  #
# 4. The analyst copies the exif directory from the remote system
#      sudo cp -r /mnt/remote/exif /tmp
#      This directory has Phil Harvey's exiftool and this script
# 5. The analyst executes this script
#      cd /tmp/exif
#      ./extractdata.sh Name fsLocation
#      Name is the name of the local computer (to distinguish it on the remote system)
#      fsLocation is the location of the unmounted(!) ntfs file system (e.g., /dev/sda2)
#
#      The script gathers the relevant artifacts and copies it to /mnt/remote/Name
#####
# This script is copied to the local machine via the network.
# It assumes that the network location to copy to has
# previously been mounted to /mnt/remote

# Ensure script call is formatted correctly
if [ "x$1x" == "xx" ] || [ "x$2x" == "xx" ]
then
    echo ./script computerName driveLocation
exit 1
fi

# Mount the local drive
sudo mkdir /mnt/local
sudo mount -t ntfs $2 /mnt/local -o ro

# Create the remote directory to dump this box into
sudo mkdir /mnt/remote/$1

# Copy the registry tables
sudo mkdir /mnt/remote/$1/registry
sudo cp /mnt/local/Windows/System32/config/* /mnt/remote/$1/registry

# Copy the prefetch files
sudo mkdir /mnt/remote/$1/prefetch
sudo cp /mnt/local/Windows/Prefetch/* /mnt/remote/$1/prefetch

# Copy the event logs
sudo mkdir /mnt/remote/$1/eventlog
sudo cp /mnt/local/Windows/System32/winevt/Logs/* /mnt/remote/$1/eventlog

```

```

# Remove space (0x20) as a file separator
SAVEIFS=$IFS
IFS=$(echo -en "\n\b")

# Create a local file for the exif information
touch /tmp/exifinfo.txt

# Create a remote users directory for user specific information
sudo mkdir /mnt/remote/$1/users

# For each user
for username in `ls /mnt/local/Users`
do
    # Create a remote directory for the user
    sudo mkdir /mnt/remote/$1/users/$username

    # Copy the user registry tables
    sudo mkdir /mnt/remote/$1/users/$username/registry
    sudo cp /mnt/local/Users/$username/NTUSER*
/mnt/remote/$1/users/$username/registry
    sudo cp /mnt/local/Users/$username/AppData/Local/Microsoft/
Windows/UsrClass* /mnt/remote/$1/users/$username/registry

    # Copy the Chrome web browsing history
    sudo mkdir /mnt/remote/$1/users/$username/chrome
    sudo cp -r /mnt/local/Users/$username/AppData/Local/Google/
Chrome/User\ Data/Default/History*
/mnt/remote/$1/users/$username/chrome

    # Copy the Internet Explorer web browsing history
    sudo mkdir /mnt/remote/$1/users/$username/ie
    sudo cp -r /mnt/local/Users/$username/AppData/Local/Microsoft/Windows/
History/* /mnt/remote/$1/users/$username/ie
    sudo cp -r /mnt/local/Users/$username/AppData/Local/Microsoft/Windows/
IEDownloadHistory/container.dat
/mnt/remote/$1/users/$username/ie/dlhistory.container.dat

    # Copy the Firefox web browsing history
    sudo mkdir /mnt/remote/$1/users/$username/firefox
    for dirname in `ls /mnt/local/Users/$username/AppData/Roaming/Mozilla/
Firefox/Profiles`
    do
        sudo mkdir /mnt/remote/$1/users/$username/firefox/$dirname
    done
done

```

```

    sudo cp /mnt/local/Users/$username/AppData/Roaming/Mozilla/Firefox/
Profiles/$dirname/places.sql*
    /mnt/remote/$1/users/$username/firefox/$dirname
done

# Generate extended information stored in the file headers (exclude Crypt
# and System Certificates because they make exiftool hang)
sudo find /mnt/local/Users/$username * -print |
grep -v Crypt | grep -v SystemCertificates >/tmp/user_list.inp
for i in `cat /tmp/user_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done

echo Finished User $username
done

echo Exited user copy

# Create a directory list of the local file system
sudo find /mnt/local * -print >/tmp/listing.inp

echo Created file list

# Create a list of file times for every file on the local file system
# Stat will only capture 3 times -- below I'm using ntfsinfo to capture all of them
# However, I wanted two files just to be on the safe side
# statinfo.txt will probably end up not being used
touch /tmp/statinfo.txt
for i in `cat /tmp/listing.inp`
do
    sudo stat "$i" >>/tmp/statinfo.txt
done

# Copy the stat information
sudo cp /tmp/statinfo.txt /mnt/remote/$1/statinfo.txt

echo Copied stat information

# Generate extended information stored in the file headers for specific file types,
# regardless of where on the local file system they may be

```



```

sudo find /mnt/local -name *.pdf -print >/tmp/pdf_list.inp
for i in `cat /tmp/pdf_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.doc* -print >/tmp/doc_list.inp
for i in `cat /tmp/doc_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.ppt* -print >/tmp/ppt_list.inp
for i in `cat /tmp/ppt_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.xls* -print >/tmp/xls_list.inp
for i in `cat /tmp/xls_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.jpg -print >/tmp/jpg_list.inp
for i in `cat /tmp/jpg_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.mpg* -print >/tmp/mpg_list.inp
for i in `cat /tmp/mpg_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done
sudo find /mnt/local -name *.wav* -print >/tmp/wav_list.inp

```

```

for i in `cat /tmp/wav_list.inp`
do
    sudo ./exiftool "$i" | sed -n -e '/Modify Date/p' -e '/Create Date/p'
-e '/Creator/p' -e '/Last Modified Time/p' -e '/File Name/p'
-e '/Directory/p' >>/tmp/exifinfo.txt
done

# Copy all the generated extended file information
sudo cp /tmp/exifinfo.txt /mnt/remote/$1/exifinfo.txt

echo Copied exif information

# Unmount the local filesystem (necessary for ntfsinfo to work)
sudo umount /mnt/local
sudo rmdir /mnt/local

# Create a list of file times for every file on the local file system
touch /tmp/ntfsinfo.txt
for i in `cat /tmp/listing.inp | sed -n -e 's/\mnt\/local\////p'`
do
    echo $i >>/tmp/ntfsinfo.txt
    sudo ntfsinfo -F $i $2 | sed -n -e '/Dumping Inode/p' -e '/User ID/p'
-e '/Security ID:/p'
-e '/Dumping attribute $FILE_NAME/,/Dumping attribute $DATA/
{/File Creation Time/p; /File Altered Time/p;
/MFT Changed Time/p; /Last Accessed Time/p;
/Namespace/p; /Filename:/p; /Parent/p}'
>>/tmp/ntfsinfo.txt
done

# Copy the ntfs information
sudo cp /tmp/ntfsinfo.txt /mnt/remote/$1/ntfsinfo.txt

# Unmount the remote file system
sudo umount /mnt/remote
sudo rmdir /mnt/remote

# Restore the file separator
IFS=$SAVEIFS

```

5.3 RegRipper Scripts

Event Messages.

```
#-----
# eventmsg
# Test RegRipper Script
#
# History
#   20131016 - created by copying then modifying the tearextract plug-in
#
# References
#
# copyright 2016 Jimmy Okolica
#-----

package eventmsg;
use strict;
use Time::Local;
use Encode;

my %config = (hive          => "System",
              osmask        => 22,
              hasShortDescr => 1,
              hasDescr      => 0,
              hasRefs       => 0,
              version       => 20161005);

my $sdate;
my $edate;
my %out_arr;

sub getConfig{return %config}

sub getShortDescr {
return "Event Message File Extraction";
}
sub getDescr{}
sub getRefs {}
sub getHive {return $config{hive};}
sub getVersion {return $config{version};}
```

```

my $VERSION = getVersion();
my $ccs = "";

#-----
# Main Routine
#-----
sub pluginmain {
my $class = shift;
my $hive = shift;
    $sdate = timelocal(0,0,0,1,0,2009);
    $edate = timelocal(59,59,23,31,11,2013);
::logMsg("Launching tearextract v.".$VERSION);
# ::rptMsg("drivers32 v.".$VERSION); # banner
# ::rptMsg("(".$config{hive}." ) ".getShortDescr()."\n"); # banner
my $reg = Parse::Win32Registry->new($hive);
my $root_key = $reg->get_root_key;
my @fname = split(/\x2f/, $reg->get_filename());
if (scalar(@fname) < 2) {
@fname = split(/\x5c/, $reg->get_filename());
}
my $root_name = @fname[scalar(@fname) - 1];
my $key_path = 'Select';
my $key;
if ($key = $root_key->get_subkey($key_path)) {
my $current = $key->get_value("Current")->get_data();
$ccs = "ControlSet00".$current;
}
# Debug Message
# ::rptMsg('Current Control Set: '.$ccs);

my @keys = $root_key->get_list_of_subkeys();

# Starting from the top, proceed through each key
foreach my $key (@keys) {

    # Debug Message
    # ::rptMsg('Hive: '.$key->get_name());

    # For each key, process all of the values
    my @vk = $key->get_list_of_values();
    if (scalar(@vk) > 0) {
        foreach my $v (@vk) {

```

```

        processvalues($root_name, $key, $v);
    }

    } # if @vk > 0

    # Then process all of the subkeys of that key
    checkSubKeys($key,0,$root_name.'/'.'$key->get_name());
}

foreach my $name (keys %out_arr) {
    ::rptMsg($out_arr{$name}{key}."\t".$out_arr{$name}{file});
    #::rptMsg("\"".$out_arr{$name}{key}."\"".\"\\t\".\"\"".$out_arr{$name}{file}."\"");
}
}

#-----
# Recursive shell for processing all levels of keys
#-----
sub checkSubKeys {
    my $key = shift;
    my $level = shift;
    my $keyname = shift;

    my @keys = $key->get_list_of_subkeys();

    if (scalar(@keys) > 0) {

        # For each key
        foreach my $skey (@keys) {

            # Process all of the values
            my @vk = $skey->get_list_of_values();
            if (scalar(@vk) > 0) {
                foreach my $v (@vk) {
                    processvalues($keyname, $skey, $v);
                }
            } # if @vk > 0

            # Process all the subkeys under that key
            checkSubKeys($skey,$level + 1,$keyname.'/'.'$skey->get_name());

        } # for each @keys
    }
}

```

```

    } # if @keys > 0
}

#-----
# Worker routine - pattern match the key/value to see if it
# needs processing and if so, process it for later output
#-----
sub processvalues {

my $keyname = shift;
my $skey = shift;
my $v = shift;

    my $val = $v->get_data;
    my $name = $keyname.'/'.'$skey->get_name().'/'.'$v->get_name;

# Filenames accessed recently via Windows Explorer
# USRCLASS/Local Settings/Software/Microsoft/Windows/CurrentVersion/Shell/BagMRU
# source code comes from regripper/shellbags.pl
if (index(uc($name), "SYSTEM/" . uc($ccs) . "/SERVICES/EVENTLOG") > -1) {
    my $name = $keyname.'/'.'$skey->get_name();
if ( (index(uc($v->get_name()), "EVENTMESSAGEFILE") > -1) || (index(uc($v->get_name()),
my $windex = "." . $name . ".";
$out_arr{$name}{file} = $v->get_data();
$out_arr{$name}{key} = $skey->get_name();
}

}

}

1;

```

Date and Time for all Registry Keys.

```

#-----
# datetime
# Test RegRipper Script
#
# History
#   20131016 - created by copying then modifying the drivers32 plug-in
#
# References
#

```

```

# copyright 2013 Jimmy Okolica
#-----
package datetime;
use strict;
use Time::Local;
use Encode;

my %config = (hive          => "Software",
              osmask        => 22,
              hasShortDescr => 1,
              hasDescr      => 0,
              hasRefs       => 0,
              version       => 20130408);

my $sdate;
my $edate;
my %csm_arr;

sub getConfig{return %config}

sub getShortDescr {
return "Retrieve all events with a last update in range";
}
sub getDescr{}
sub getRefs {}
sub getHive {return $config{hive};}
sub getVersion {return $config{version};}

my $VERSION = getVersion();
my $ccs = "";

#-----
# Main Routine
#-----
sub pluginmain {
my $class = shift;
my $hive = shift;
    $sdate = timelocal(0,25,9,7,7,2015);
    $edate = timelocal(59,59,23,7,7,2017);
    ::logMsg("Launching tearextract v.".$VERSION);
    # ::rptMsg("drivers32 v.".$VERSION); # banner
    # ::rptMsg("(".$config{hive}.") ".getShortDescr()."\n"); # banner
my $reg = Parse::Win32Registry->new($hive);

```

```

my $root_key = $reg->get_root_key;
my @fname = split(/\x2f/, $reg->get_filename());
if (scalar(@fname) < 2) {
    @fname = split(/\x5c/, $reg->get_filename());
}
my $root_name = @fname[scalar(@fname) - 1];
my $key_path = 'Select';
my $key;
if ($key = $root_key->get_subkey($key_path)) {
    my $current = $key->get_value("Current")->get_data();
    $ccs = "ControlSet00".$current;
}
# Debug Message
# ::rptMsg('Current Control Set: '.$ccs);

my @keys = $root_key->get_list_of_subkeys();

# Starting from the top, proceed through each key
foreach my $key (@keys) {

    # Debug Message
    # ::rptMsg('Hive: '.$key->get_name());

    # For each key, process all of the values
    my @vk = $key->get_list_of_values();
    if (scalar(@vk) > 0) {
        foreach my $v (@vk) {
            processvalues($root_name, $key, $v, $sdate, $edate);
        }
        } # if @vk > 0

    # Then process all of the subkeys of that key
    checkSubKeys($key, 0, $root_name.'/'.'$key->get_name(), $sdate, $edate);
}

foreach my $x (keys %csm_arr) {
    my $files = (split(/\00\00/, $csm_arr{$x}{value}, 2))[0];
    $files =~ s/\00//g;
    ::rptMsg("**TIME DATA ONLY**/" . $csm_arr{$x}{name} . "/" . $x .
    "\t" . $x . "~" . $files . "\t" . $csm_arr{$x}{time});
}

```



```

}

#-----
# Recursive shell for processing all levels of keys
#-----
sub checkSubKeys {
    my $key = shift;
    my $level = shift;
    my $keyname = shift;
    my $sdate = shift;
    my $edate = shift;

    my @keys = $key->get_list_of_subkeys();

    if (scalar(@keys) > 0) {

        # For each key
        foreach my $skey (@keys) {

            # Process all of the values
            my @vk = $skey->get_list_of_values();
            if (scalar(@vk) > 0) {
                foreach my $v (@vk) {
                    processvalues($keyname, $skey, $v, $sdate, $edate);
                }
            } # if @vk > 0

            # Process all the subkeys under that key
            checkSubKeys($skey,$level + 1,$keyname.'/'.'$skey->get_name()', $sdate, $edate);

        } # for each @keys
    } # if @keys > 0
}

#-----
# Worker routine - pattern match the key/value to see if it
# needs processing and if so, process it for later output
#-----
sub processvalues {

    my $keyname = shift;

```

```

my $skey = shift;
my $v = shift;
my $sdate = shift;
my $edate = shift;

    my $val = $v->get_data;
    my $name = $keyname.'/'.'$skey->get_name().'/'.'$v->get_name;
my $time = $skey->get_timestamp();

if (($sdate <= $time) && ($time <= $edate)) {
    my $name = $keyname.'/'.'$skey->get_name();
$csm_arr{$v->get_name()}{name} = $name;
$csm_arr{$v->get_name()}{value} = $v->get_data();
$csm_arr{$v->get_name()}{time} = $skey->get_timestamp();
}
}
1;

```

Primary Registry Extraction Script.

```

#-----
# tearextract
# Test RegRipper Script
#
# History
#   20131016 - created by copying then modifying the drivers32 plug-in
#
# References
#
# copyright 2013 Jimmy Okolica
#-----

package tearextract;
use strict;
use Time::Local;
use Encode;

my %config = (hive          => "Software",
              osmask        => 22,
              hasShortDescr => 1,
              hasDescr      => 0,
              hasRefs       => 0,
              version       => 20130408);

my $sdate;

```

```

my $edate;
my %out_arr;
my @wwl_order;
my %wwl_arr;
my @lvp_order;
my %lvp_arr;
my @lvp2_order;
my %lvp2_arr;
my @csm_order;
my %csm_arr;
my @ff_order;
my %ff_arr;
my %rd_order;
my %rd_arr;
my %osp_order;
my %osp_arr;
my %bags_order;
my %bags_arr;

my %keyvalue;
my %item;

my %cp_guids = ("{bb64f8a7-bee7-4e1a-ab8d-7d8273f7fdb6}" => "Action Center",
    "{7a979262-40ce-46ff-ae4e-7884ac3b6136}" => "Add Hardware",
    "{d20ea4e1-3957-11d2-a40b-0c5020524153}" => "Administrative Tools",
    "{9c60de1e-e5fc-40f4-a487-460851a8d915}" => "AutoPlay",
    "{b98a2bea-7d42-4558-8bd1-832f41bac6fd}" => "Backup and Restore Center",
    "{0142e4d0-fb7a-11dc-ba4a-000ffe7ab428}" => "Biometric Devices",
    "{d9ef8727-cac2-4e60-809e-86f80a666c91}" => "BitLocker Drive Encryption",
    "{b2c761c6-29bc-4f19-9251-e6195265baf1}" => "Color Management",
    "{1206f5f1-0569-412c-8fec-3204630dfb70}" => "Credential Manager",
    "{e2e7934b-dce5-43c4-9576-7fe4f75e7480}" => "Date and Time",
    "{00c6d95f-329c-409a-81d7-c46c66ea7f33}" => "Default Location",
    "{17cd9488-1228-4b2f-88ce-4298e93e0966}" => "Default Programs",
    "{37efd44d-ef8d-41b1-940d-96973a50e9e0}" => "Desktop Gadgets",
    "{74246bfc-4c96-11d0-abef-0020af6b0b7a}" => "Device Manager",
    "{a8a91a66-3a7d-4424-8d24-04e180695c7a}" => "Devices and Printers",
    "{c555438b-3c23-4769-a71f-b6d3d9b6053a}" => "Display",
    "{d555645e-d4f8-4c29-a827-d93c859c4f2a}" => "Ease of Access Center",
    "{6dfd7c5c-2451-11d3-a299-00c04f8ef6af}" => "Folder Options",
    "{93412589-74d4-4e4e-ad0e-e0cb621440fd}" => "Fonts",
    "{259ef4b1-e6c9-4176-b574-481532c9bce8}" => "Game Controllers",
    "{15eae92e-f17a-4431-9f28-805e482dafd4}" => "Get Programs",

```

"{cb1b7f8c-c50a-4176-b604-9e24dee8d4d1}" => "Getting Started",
 "{67ca7650-96e6-4fdd-bb43-a8e774f73a57}" => "HomeGroup",
 "{87d66a43-7b11-4a28-9811-c86ee395acf7}" => "Indexing Options",
 "{a0275511-0e86-4eca-97c2-ecd8f1221d08}" => "Infrared",
 "{a3dd4f92-658a-410f-84fd-6fbbbf2fffe}" => "Internet Options",
 "{a304259d-52b8-4526-8b1a-a1d6cecc8243}" => "iSCSI Initiator",
 "{725be8f7-668e-4c7b-8f90-46bdb0936430}" => "Keyboard",
 "{e9950154-c418-419e-a90a-20c5287ae24b}" => "Location and Other Sensors",
 "{1fa9085f-25a2-489b-85d4-86326eedcd87}" => "Manage Wireless Networks",
 "{6c8eec18-8d75-41b2-a177-8831d59d2d50}" => "Mouse",
 "{7007acc7-3202-11d1-aad2-00805fc1270e}" => "Network Connections",
 "{8e908fc9-becc-40f6-915b-f4ca0e70d03d}" => "Network and Sharing Center",
 "{05d7b0f4-2121-4eff-bf6b-ed3f69b894d9}" => "Notification Area Icons",
 "{d24f75aa-4f2b-4d07-a3c4-469b3d9030c4}" => "Offline Files",
 "{96ae8d84-a250-4520-95a5-a47a7e3c548b}" => "Parental Controls",
 "{f82df8f7-8b9f-442e-a48c-818ea735ff9b}" => "Pen and Input Devices",
 "{5224f545-a443-4859-ba23-7b5a95bdc8ef}" => "People Near Me",
 "{78f3955e-3b90-4184-bd14-5397c15f1efc}" => "Performance Information and Tools",
 "{ed834ed6-4b5a-4bfe-8f11-a626dcb6a921}" => "Personalization",
 "{40419485-c444-4567-851a-2dd7bfa1684d}" => "Phone and Modem",
 "{025a5937-a6be-4686-a844-36fe4bec8b6d}" => "Power Options",
 "{2227a280-3aea-1069-a2de-08002b30309d}" => "Printers",
 "{fcfeecae-ee1b-4849-ae50-685dcf7717ec}" => "Problem Reports and Solutions",
 "{7b81be6a-ce2b-4676-a29e-eb907a5126c5}" => "Programs and Features",
 "{9fe63afd-59cf-4419-9775-abcc3849f861}" => "Recovery",
 "{62d8ed13-c9d0-4ce8-a914-47dd628fb1b0}" => "Regional and Language Options",
 "{241d7c96-f8bf-4f85-b01f-e2b043341a4b}" => "RemoteApp and Desktop Connections",
 "{00f2886f-cd64-4fc9-8ec5-30ef6cdbe8c3}" => "Scanners and Cameras",
 "{f2ddfc82-8f12-4cdd-b7dc-d4fe1425aa4d}" => "Sound",
 "{58e3c745-d971-4081-9034-86e34b30836a}" => "Speech Recognition Options",
 "{9c73f5e5-7ae7-4e32-a8e8-8d23b85255bf}" => "Sync Center",
 "{bb06c0e4-d293-4f75-8a90-cb05b6477eee}" => "System",
 "{80f3f1d5-feca-45f3-bc32-752c152e456e}" => "Tablet PC Settings",
 "{0df44eaa-ff21-4412-828e-260a8728e7f1}" => "Taskbar and Start Menu",
 "{d17d1d6d-cc3f-4815-8fe3-607e7d5d10b3}" => "Text to Speech",
 "{c58c4893-3be0-4b45-abb5-a63e4b8c8651}" => "Troubleshooting",
 "{60632754-c523-4b62-b45c-4172da012619}" => "User Accounts",
 "{be122a0e-4503-11da-8bde-f66bad1e3f3a}" => "Windows Anytime Upgrade",
 "{78cb147a-98ea-4aa6-b0df-c8681f69341c}" => "Windows CardSpace",
 "{d8559eb9-20c0-410e-beda-7ed416aecc2a}" => "Windows Defender",
 "{4026492f-2f69-46b8-b9bf-5654fc07e423}" => "Windows Firewall",
 "{3e7efb4c-faf1-453d-89eb-56026875ef90}" => "Windows Marketplace",
 "{5ea4f148-308c-46d7-98a9-49041b1dd468}" => "Windows Mobility Center",

```

"{e95a4861-d57a-4be1-ad0f-35267e261739}" => "Windows SideShow",
"{36eef7db-88ad-4e81-ad49-0e313f0c35f8}" => "Windows Update");

my %folder_types = ("{724ef170-a42d-4fef-9f26-b60e846fba4f}" => "Administrative Tools",
"{d0384e7d-bac3-4797-8f14-cba229b392b5}" => "Common Administrative Tools",
"{de974d24-d9c6-4d3e-bf91-f4455120b917}" => "Common Files",
"{c1bae2d0-10df-4334-bedd-7aa20b227a9d}" => "Common OEM Links",
"{5399e694-6ce5-4d6c-8fce-1d8870fdcba0}" => "Control Panel",
"{1ac14e77-02e7-4e5d-b744-2eb1ae5198b7}" => "CSIDL_SYSTEM",
"{b4bfcc3a-db2c-424c-b029-7fe99a87c641}" => "Desktop",
"{7b0db17d-9cd2-4a93-9733-46cc89022e7c}" => "Documents Library",
"{fdd39ad0-238f-46af-adb4-6c85480369c7}" => "Documents",
"{374de290-123f-4565-9164-39c4925e467b}" => "Downloads",
"{de61d971-5ebc-4f02-a3a9-6c82895e5c04}" => "Get Programs",
"{a305ce99-f527-492b-8b1a-7e76fa98d6e4}" => "Installed Updates",
"{871c5380-42a0-1069-a2ea-08002b30309d}" => "Internet Explorer (Homepage)",
"{031e4825-7b94-4dc3-b131-e946b44c8dd5}" => "Libraries",
"{4bd8d571-6d19-48d3-be97-422220080e43}" => "Music",
"{20d04fe0-3aea-1069-a2d8-08002b30309d}" => "My Computer",
"{450d8fba-ad25-11d0-98a8-0800361b1103}" => "My Documents",
"{ed228fdf-9ea8-4870-83b1-96b02cfe0d52}" => "My Games",
"{208d2c60-3aea-1069-a2d7-08002b30309d}" => "My Network Places",
"{f02c1a0d-be21-4350-88b0-7367fc96ef3c}" => "Network",
"{33e28130-4e1e-4676-835a-98395c3bc3bb}" => "Pictures",
"{a990ae9f-a03b-4e80-94bc-9912d7504104}" => "Pictures",
"{7c5a40ef-a0fb-4bfc-874a-c0f2e0b9fa8e}" => "Program Files (x86)",
"{905e63b6-c1bf-494e-b29c-65b732d3d21a}" => "Program Files",
"{df7266ac-9274-4867-8d55-3bd661de872d}" => "Programs and Features",
"{3214fab5-9757-4298-bb61-92a9deaa44ff}" => "Public Music",
"{b6ebfb86-6907-413c-9af7-4fc2abf07cc5}" => "Public Pictures",
"{2400183a-6185-49fb-a2d8-4a392a602ba3}" => "Public Videos",
"{4336a54d-38b-4685-ab02-99bb52d3fb8b}" => "Public",
"{491e922f-5643-4af4-a7eb-4e7a138d8174}" => "Public",
"{dfdf76a2-c82a-4d63-906a-5644ac457385}" => "Public",
"{645ff040-5081-101b-9f08-00aa002f954e}" => "Recycle Bin",
"{d65231b0-b2f1-4857-a4ce-a8e7c6ea7d27}" => "System32 (x86)",
"{9e52ab10-f80d-49df-acb8-4330f5687855}" => "Temporary Burn Folder",
"{f3ce0f7c-4901-4acc-8648-d5d44b04ef8f}" => "Users Files",
"{59031a47-3f72-44a7-89c5-5595fe6b30ee}" => "Users",
"{f38bf404-1d43-42f2-9305-67de0b28fc23}" => "Windows");

```

```

sub getConfig{return %config}

sub getShortDescr {
return "Temporal Event Abstraction and Reconstruction";
}
sub getDescr{}
sub getRefs {}
sub getHive {return $config{hive};}
sub getVersion {return $config{version};}

my $VERSION = getVersion();
my $ccs = "";

#-----
# Main Routine
#-----
sub pluginmain {
my $class = shift;
my $hive = shift;
    $sdate = timelocal(0,0,0,1,0,2009);
    $edate = timelocal(59,59,23,31,11,2013);
::logMsg("Launching tearextract v.".$VERSION);
# ::rptMsg("drivers32 v.".$VERSION); # banner
# ::rptMsg("(".$config{hive}.") ".getShortDescr()."\n"); # banner
my $reg = Parse::Win32Registry->new($hive);
my $root_key = $reg->get_root_key;
my @fname = split(/\x2f/,$reg->get_filename());
if (scalar(@fname) < 2) {
@fname = split(/\x5c/,$reg->get_filename());
}
my $root_name = @fname[scalar(@fname) - 1];
my $key_path = 'Select';
my $key;
if ($key = $root_key->get_subkey($key_path)) {
my $current = $key->get_value("Current")->get_data();
$ccs = "ControlSet00".$current;
}
# Debug Message
# ::rptMsg('Current Control Set: '.$ccs);

my @keys = $root_key->get_list_of_subkeys();

```

```

# Starting from the top, proceed through each key
foreach my $key (@keys) {

    # Debug Message
    # ::rptMsg('Hive: ' . $key->get_name());

    # For each key, process all of the values
    my @vk = $key->get_list_of_values();
    if (scalar(@vk) > 0) {
        foreach my $v (@vk) {
            processvalues($root_name, $key, $v);
        }
        } # if @vk > 0

    # Then process all of the subkeys of that key
    checkSubKeys($key,0,$root_name.'/' . $key->get_name());
}

foreach my $x2 (keys %bags_order) {
my $wlist = $bags_order{$x2}{list};
    my @wlist = unpack("V*", $wlist);
next if ( ($wlist[0] == 0xffff) || ($wlist[0] == 0xffffffff));
foreach my $xkey (@wlist) {
my $windex = $x2.$xkey;
next unless (exists $bags_arr{$windex}{name});
::rptMsg($bags_arr{$windex}{name} . "/" . $xkey . "\t" . $xkey . "~" .
$bags_arr{$windex}{value} . "\t" . $bags_arr{$windex}{time});
}
}

foreach my $x2 (keys %osp_order) {
my $wlist = $osp_order{$x2}{list};
    my @wlist = unpack("V*", $wlist);
foreach my $xkey (@wlist) {
my $windex = $x2.$xkey;
my $str = parseShellItem($osp_arr{$windex}{value});
::rptMsg($osp_arr{$windex}{name} . "/" . $xkey . "\t" . $xkey . "~" .
$str . "\t" . $osp_arr{$windex}{time});
}
}

```

```

foreach my $x (@lvp_order) {
my ($file,$shell) = split(/\00\00/,$lvp_arr{$x}{value},2);
$file =~ s/\00//g;
$shell =~ s/^\00//;
my $str = parseShellItem($shell);
::rptMsg($lvp_arr{$x}{name}."/".$x."\t".$x."~".$file."~".$str."\t".
$lvp_arr{$x}{time});
}

foreach my $x (@lvp2_order) {
my ($file,$shell) = split(/\00\00/,$lvp2_arr{$x}{value},2);
$file =~ s/\00//g;
$shell =~ s/^\00//;
my $str = parseShellItem($shell);
::rptMsg($lvp2_arr{$x}{name}."/".$x."\t".$x."~".$file."~".$str."\t".
$lvp2_arr{$x}{time});
}

foreach my $x (@ff_order) {
my @files = split(/\00\00/,$ff_arr{$x}{value});
if (scalar(@files) == 1) {
$files[0] =~ s/\00//g;
::rptMsg($ff_arr{$x}{name}."\t".$files[0]."\t".$ff_arr{$x}{time});
} elsif (scalar(@files) > 1) {
my @files2;
foreach my $file (@files) {
$file =~ s/\00//g;
push(@files2,$file);
}
::rptMsg($ff_arr{$x}{name}."/".$x."\t".$x."~".join('~',@files2)."\t".
$ff_arr{$x}{time});
}
}

foreach my $x (@csm_order) {
my $files = (split(/\00\00/,$csm_arr{$x}{value},2))[0];
$files =~ s/\00//g;
::rptMsg($csm_arr{$x}{name}."/".$x."\t".$x."~".$files."\t".$csm_arr{$x}{time});
}

foreach my $x2 (keys %rd_order) {
my $wlist = $rd_order{$x2}{list};

```



```

        my @wlist = unpack("V*", $wlist);
foreach my $xkey (@wlist) {
my $windex = $x2.$xkey;
::rptMsg($rd_arr{$windex}{name}."/".$xkey."\t".$xkey."~".
$rd_arr{$windex}{value}."\t".$rd_arr{$windex}{time});
}
}

foreach my $x (@wwl_order) {
::rptMsg($wwl_arr{$x}{name}."/".$x."\t".$x."~".$wwl_arr{$x}{value}."\t".
$wwl_arr{$x}{time});
}

foreach my $name (keys %out_arr) {
::rptMsg($name."\t".$out_arr{$name}{value}."\t".$out_arr{$name}{time});
}
}

#-----
# Recursive shell for processing all levels of keys
#-----
sub checkSubKeys {
    my $key = shift;
    my $level = shift;
    my $keyname = shift;

    my @keys = $key->get_list_of_subkeys();

    if (scalar(@keys) > 0) {

        # For each key
        foreach my $skey (@keys) {

            # Process all of the values
            my @vk = $skey->get_list_of_values();
            if (scalar(@vk) > 0) {
                foreach my $v (@vk) {
processvalues($keyname, $skey, $v);
                }
            } # if @vk > 0

            # Process all the subkeys under that key

```

```

        checkSubKeys($skey,$level + 1,$keyname.'/'.'$skey->get_name());

    } # for each @keys
} # if @keys > 0
}

#-----
# Worker routine - pattern match the key/value to see if it
# needs processing and if so, process it for later output
#-----
sub processvalues {

my $keyname = shift;
my $skey = shift;
my $v = shift;

    my $val = $v->get_data;
    my $name = $keyname.'/'.'$skey->get_name().'/'.'$v->get_name;

# Filenames accessed recently via Windows Explorer
# USRCLASS/Local Settings/Software/Microsoft/Windows/CurrentVersion/Shell/BagMRU
# source code comes from regripper/shellbags.pl
if (index(uc($name), "SOFTWARE/MICROSOFT/WINDOWS/SHELL/BAGMRU") > -1) {
    my $name = $keyname.'/'.'$skey->get_name();
    if (index(uc($v->get_name()),"MRULISTEX") > -1) {
my $windex = ".$name.";
$bags_order{$windex}{list} = $v->get_data();
    }

        elsif (index(uc($v->get_name()),"NODESLOT") == -1) {
my $oname = $name.'/'.'$v->get_name();
my $windex = ".$name.". ".$v->get_name();
my $type = unpack("C", substr($v->get_data(),2,1));
my $is_zip = (exists $keyvalue{$name}{iszipsubfolder} &&
$keyvalue{$name}{iszipsubfolder} == 1) ? 1 : 0;
if ($is_zip) {
%item = parseZipSubFolderItem($v->get_data());
$item{iszipsubfolder} = 1;
} elsif ($type == 0x00) { # Variable/Property Sheet
%item = parseVariableEntry($v->get_data());
} elsif ($type == 0x01) { #
%item = parse01ShellItem($v->get_data());
} elsif ($type == 0x1f) { # System Folder
%item = parseSystemFolderEntry($v->get_data());

```

```

} elseif ($type == 0x2e) { # Device
$item = parseDeviceEntry($v->get_data());
} elseif ($type == 0x2f) { # Device
$item = parseDriveEntry($v->get_data());
} elseif ( ($type == 0xc3) || ($type == 0x41) || ($type == 0x42) ||
($type == 0x46) || ($type == 0x47)) { # Network Stuff
$item = parseNetworkEntry2($v->get_data());
} elseif ( ($type == 0x31) || ($type == 0x32) || ($type == 0xb1) ||
($type == 0x74)) { # Folder or Zip File
$item = parseFolderEntry($v->get_data());
} elseif ($type == 0x35) { #
$item = parseFolderEntry2($v->get_data());
} elseif ($type == 0x71) { # Control Panel
$item = parseControlPanelEntry($v->get_data());
} elseif ($type == 0x61) { # URI type
$item = parseURIEntry($v->get_data());
}
else {
$item{name} = sprintf "Unknown Type (0x%x)", $type;
}

if ($item{name} =~ m/\.zip$/ && $type == 0x32) {
$item{zipsubfolder} = 1;
}

my ($m,$a,$c,$o,$s);
(exists $item{mtime_str} && $item{mtime_str} ne "0") ? ($m =
"~LastModified:".$item{mtime_str}) : ($m = "");
(exists $item{atime_str} && $item{atime_str} ne "0") ? ($a =
"~LastAccessed:".$item{atime_str}) : ($a = "");
(exists $item{ctime_str} && $item{ctime_str} ne "0") ? ($c =
"~Created:".$item{ctime_str}) : ($c = "");
(exists $item{datetime} && $item{datetime} ne "N/A") ? ($o =
"~LastAccessed:".$item{datetime}) : ($o = "");
(exists $item{filesize} ) ? ($s = "~Filesize:".$item{filesize}) : ($s = "");
my $parent_name = (exists $keyvalue{$name}{name}) ?
$keyvalue{$name}{name} : "Desktop";

$bags_arr{$swindex}{name} = $name;
$bags_arr{$swindex}{value} = $parent_name."/". $item{name}.$c.$a.$m.$o.$s."~";
$bags_arr{$swindex}{time} = $skey->get_timestamp();

$keyvalue{$soname}{name} = $parent_name."/". $item{name};

```

```

$keyvalue{$oname}{iszipsubfolder} = ((exists $item{zipsubfolder} &&
$item{zipsubfolder} == 1) ? 1 : $is_zip);
$keyvalue{$oname}{mtime} = ((exists $item{mtime} ) ? $item{mtime} : "");
$keyvalue{$oname}{atime} = ((exists $item{atime} ) ? $item{atime} : "");
$keyvalue{$oname}{ctime} = ((exists $item{ctime} ) ? $item{ctime} : "");
$keyvalue{$oname}{datetime} = ((exists $item{datetime} ) ? $item{datetime} : "");
$keyvalue{$oname}{filesize} = ((exists $item{filesize} ) ? $item{filesize} : "");
}

}

# Filenames accessed recently via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
# source code comes from regripper/comdlg32.pl
elsif (index(uc($name), "CURRENTVERSION/EXPLORER/COMDLG32/OPENSAREPIDLMRU") > -1) {
    my $name = $keyname.'/'.'$key->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        my $windex = ".$key->get_name.";
        $osp_order{$windex}{list} = $v->get_data();
    }
    else {
        my $windex = ".$key->get_name()."."$v->get_name();
        $osp_arr{$windex}{name} = $name;
        $osp_arr{$windex}{value} = $v->get_data();
        $osp_arr{$windex}{time} = $key->get_timestamp();
    }
}

# Filenames accessed recently via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
# source code comes from regripper/comdlg32.pl
elsif (index(uc($name), "CURRENTVERSION/EXPLORER/COMDLG32/LASTVISITEDPIDLMRULEGACY") > -1) {
    my $name = $keyname.'/'.'$key->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        @lv2_order = unpack("V*", $v->get_data());
    }
    else {
        $lv2_arr{$v->get_name(){name} = $name;
        $lv2_arr{$v->get_name(){value} = $v->get_data();
        $lv2_arr{$v->get_name(){time} = $key->get_timestamp();
    }
}

```

```

# Filenames accessed recently via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
# source code comes from regripper/comdlg32.pl
elseif (index(uc($name), "CURRENTVERSION/EXPLORER/COMDLG32/LASTVISITEDPIDLMRU") > -1) {
    my $name = $keyname.'/'.$skey->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        @lvp_order = unpack("V*", $v->get_data());
    }
    else {
        $lvp_arr{$v->get_name()}{name} = $name;
        $lvp_arr{$v->get_name()}{value} = $v->get_data();
        $lvp_arr{$v->get_name()}{time} = $skey->get_timestamp();
    }
}

# Filenames accessed recently via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
# source code comes from regripper/comdlg32.pl
elseif (index(uc($name), "CURRENTVERSION/EXPLORER/COMDLG32/FIRSTFOLDER") > -1) {
    my $name = $keyname.'/'.$skey->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        @ff_order = unpack("V*", $v->get_data());
    }
    else {
        $ff_arr{$v->get_name()}{name} = $name;
        $ff_arr{$v->get_name()}{value} = $v->get_data();
        $ff_arr{$v->get_name()}{time} = $skey->get_timestamp();
    }
}

# Filenames accessed recently via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/ComDlg32
# source code comes from regripper/comdlg32.pl
elseif (index(uc($name), "CURRENTVERSION/EXPLORER/COMDLG32/CIDSIZEMRU") > -1) {
    my $name = $keyname.'/'.$skey->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        @csm_order = unpack("V*", $v->get_data());
    }
    else {
        $csm_arr{$v->get_name()}{name} = $name;
        $csm_arr{$v->get_name()}{value} = $v->get_data();
        $csm_arr{$v->get_name()}{time} = $skey->get_timestamp();
    }
}

```

```

}

# Applications executed via Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/UserAssist
# source code comes from regripper/userassist.pl
elseif ( (index(uc($name), "CURRENTVERSION/EXPLORER/USERASSIST") > -1) &&
(index(uc($skey->get_name()), "COUNT") > -1) ) {
my $data = $v->get_data();
my $vname = $v->get_name();
my $hrzr = "HRZR";
if (length($data) == 16) { # Windows XP/ 2003/ Vista/ 2008
my ($session, $count, $val1, $val2) = unpack("V*", $data);
if ($val2 != 0) {
my $time_value = ::getTime($val1, $val2);
if ($vname =~ m/^\$hrzr/) {
$vname =~ tr/N-ZA-Mn-za-m/A-Za-z/;
}
if ($count > 5) {
$count = $count - 5;
}
$out_arr{$name}{value} = "Value:". $vname. " Count:". $count. " Time:".
gmtime($time_value);
$out_arr{$name}{time} = $skey->get_timestamp();
}
} elseif (length($data) == 72) { # Windows 7
$vname =~ tr/N-ZA-Mn-za-m/A-Za-z/;
my $count = unpack("V", substr($data, 4, 4));
my @t = unpack("VV", substr($data, 60, 8));
next if ( ($t[0] == 0) && ($t[1] == 0));
my $time_value = ::getTime($t[0], $t[1]);
$out_arr{$name}{value} = "Value:". $vname. " Count:". $count. " Time:".
gmtime($time_value);
$out_arr{$name}{time} = $skey->get_timestamp();
}
}

# Recently accessed documents in Windows Explorer by type
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/RecentDocs
# source code comes from regripper/recentdocs.pl
elseif (index(uc($name), "CURRENTVERSION/EXPLORER/RECENTDOCS") > -1) {
my $name = $keyname.'/'. $skey->get_name();
if (index(uc($v->get_name()), "MRULISTEX") > -1) {

```

```

my $windex = ".$skey->get_name.";
$rd_arr{$windex}{list} = $v->get_data();
}
else {
my $windex = ".$skey->get_name".$v->get_name();
my $wval = $v->get_data();
$wval = decode("ucs-2le", $wval);
my $file = (split(/\00/, $wval))[0];
$rd_arr{$windex}{name} = $name;
$rd_arr{$windex}{value} = $file;
$rd_arr{$windex}{time} = $skey->get_timestamp();
}
}

# MRUs of search words used in Windows Explorer
# NTUSER/Software/Microsoft/Windows/CurrentVersion/Explorer/WordWheelQuery
# source code comes from regripper/wordwheelquery.pl
elsif (index(uc($name), "CURRENTVERSION/EXPLORER/WORDWHEELQUERY") > -1) {
    my $name = $keyname.'/'.$skey->get_name();
    if (index(uc($v->get_name()), "MRULISTEX") > -1) {
        @wvl_order = unpack("V*", $v->get_data());
    }
    else {
        my $wval = $v->get_data();
        $wval =~ s/\00//g;
        $wvl_arr{$v->get_name(){name} = $name;
        $wvl_arr{$v->get_name(){value} = $wval;
        $wvl_arr{$v->get_name(){time} = $skey->get_timestamp();
    }
}

# Last time a network was created - name, date created/connected, type, and managed
# SOFTWARE/Microsoft/Windows NT/CurrentVersion/NetworkList/Profiles
# source code comes from regripper/networklist.pl
elsif ( (index(uc($name), "NETWORKLIST/PROFILES") > -1) ) {
    my $name = $keyname.'/'.$skey->get_name();
    if (index(uc($v->get_name()), "DESCRIPTION") > -1) {
        $out_arr{$name}{value} = $out_arr{$name}{value}."NAME:".$v->get_data()."\n";
        if (( !exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
            $out_arr{$name}{time} = $skey->get_timestamp();
        }
    }
    elsif (index(uc($v->get_name()), "MANAGED") > -1) {

```

```

$out_arr{$name}{value} = $out_arr{$name}{value}."MANAGED:". $v->get_data()."~";
if ( (!exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
    } elseif (index(uc($v->get_name()), "NAMETYPE") > -1) {
if ($v->get_data() == 0x47) {
$out_arr{$name}{value} = $out_arr{$name}{value}."Wireless~";
    if ( (!exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
} elseif ($v->get_data() == 0x06) {
$out_arr{$name}{value} = $out_arr{$name}{value}."Wired~";
    if (( !exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
} elseif ($v->get_data() == 0x17) {
$out_arr{$name}{value} = $out_arr{$name}{value}."Broadband(3g)~";
    if (( !exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
}

    } elseif (index(uc($v->get_name()), "DATECREATED") > -1) {
$out_arr{$name}{value} = $out_arr{$name}{value}.
"DATECREATED:".parsedate128_1($v->get_data())."~";
    if (( !exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
    } elseif (index(uc($v->get_name()), "DATELASTCONNECTED") > -1) {
$out_arr{$name}{value} = $out_arr{$name}{value}.
"DATELASTCONNECTED:".parsedate128_1($v->get_data())."~";
    if ( (!exists $out_arr{$name}{time} ) || ($out_arr{$name}{time} == 0)) {
$out_arr{$name}{time} = $skey->get_timestamp();
}
}
}
}

# Last time a device was connected - device name and possible a MAC address
# SYSTEM/MountedDevices
# source code comes from regripper/mountdev2.pl
elseif (index(uc($name), "MOUNTEDDEVICES") > -1) {
if ( (length($v->get_data()) > 12) &&
    ($v->get_name() =~ m/^\??\??\Volume/) ) {

```



```

my $id = $v->get_name();
$id =~ s/^\\(?:\?\\Volume{\\;
$id =~ s/}$//;
my $m = (split(/-/,$id,5))[4];
$m = uc($m);
$m = join(':',unpack("(A2)*",$m));
my $wval = $v->get_data();
$wval = decode("ucs-2le", $wval);
my ($dev, $sn) = (split(/#/, $wval))[1,2];
$out_arr{$name}{value} = $dev." ".$sn." GUID:{".$id."} MAC:{".$m;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg( $name."\t"."{".$id."} ".$wval." MAC:{".$m."\t".$skey->get_timestamp());
}
}

# Last time a device was connected - device name and possibly MAC address
# NTUSER.DATA/Software/Microsoft/Windows/CurrentVersion/Explorer/MountPoints2
# source code comes from regripper/mp2.pl
elsif ( (index(uc($name), "MOUNTPOINTS2") > -1) ) {
# && (index(uc($v->get_name()), "CLSID") > -1) ) {
my $tststr="";
if (index($name,'{') > 0) {
my $tststr2 = substr($name,index($name,'{'));
$tststr = join('','{"',substr($tststr2,1,index($tststr2,'/') - 1));
} elsif (index($name,'#') > 0) {
my $tststr2 = substr($name,index($name,'#'));
$tststr = join('','"#',substr($tststr2,1,index($tststr2,'/') - 1));
}
if ($tststr =~ m/{/) {
my $m = (split(/-/,$tststr,5))[4];
$m =~ s/}$//;
$m = uc($m);
$m = join(':',unpack("(A2)*",$m));
$out_arr{$name}{value} = "N/A N/A GUID:{".$tststr." MAC:{".$m;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg( $name."\t".$tststr." ".MAC:{".$m."\t".gmtime($skey->get_timestamp());
}
elsif ($tststr =~ m/{[A-Z]/) {
$out_arr{$name}{value} = "N/A N/A GUID:{".$tststr." DRIVE";
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message

```

```

# ::rptMsg( $name."\t".$tststr." ". "DRIVE"." \t".gmtime($skey->get_timestamp()));
}
elseif ($tststr =~ m/^#/) {
$out_arr{$name}{value} = "N/A N/A GUID:".$tststr." REMOTE";
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg( $name."\t".$tststr." ". "REMOTE"." \t".gmtime($skey->get_timestamp()));
}
}

# Last time a usb device was connected - device name and serial #
# SYSTEM/{Current Control Set}/Enum/USB
# source code comes from regripper/usbdevices.pl
# (I think I'm sending device & SN)
elseif ( (index(uc($name), "ENUM/USB") > -1) &&
        (index(uc($v->get_name()), "SERVICE") > -1) &&
        (index(uc($v->get_data()), "USBSTOR") > -1) ) {
my $sname = (split(/\x2f/, $name))[4];
$out_arr{$name}{value} = $sname." ".$skey->get_name();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$sname." ".$skey->get_name()."\t".gmtime($skey->get_timestamp()))
}

# Last time a usb device was connected - device name and serial #
# SYSTEM/{Current Control Set}/Control/DeviceClasses/
#          53F56307-B6BF-11D0-94F2-00A0C91EFB8B
# source code comes from regripper/devclass.pl
elseif ( (index(uc($name), "53F56307-B6BF-11D0-94F2-00A0C91EFB8B") > -1) &&
        (index(uc($name), "USBSTOR") > -1) &&
        (index(uc($v->get_name()), "DEVICEINSTANCE") > -1) ) {
my ($dev, $sn) = (split(/#/,$name))[4,5];
$out_arr{$name}{value} = $dev." ".$sn;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$dev." ".$sn."\t".gmtime($skey->get_timestamp()));
}

# Last time a usb device was connected - device name and serial #
# SYSTEM/{Current Control Set}/Control/DeviceClasses/
#          53F56307-B6BF-11D0-94F2-00A0C91EFB8B
# source code comes from regripper/devclass.pl
elseif ( (index(uc($name), "53F5630D-B6BF-11D0-94F2-00A0C91EFB8B") > -1) &&

```

```

        (index(uc($name), "REMOVABLEMEDIA") > -1) &&
        (index(uc($v->get_name()), "DEVICEINSTANCE") > -1) ) {
my $ppi = (split(/#/, $name))[5];
$out_arr{$name}{value} = $ppi;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$ppi."\t".gmtime($skey->get_timestamp()));
}

# Last time a usb device was connected - includes friendly name, device name,
# and serial number
# SOFTWARE/Microsoft/Windows Portable Devices/Devices/Friendly Name
# source code comes from regripper/removdev.pl
elsif ( (index(uc($name), "MICROSOFT/WINDOWS PORTABLE DEVICES/DEVICES") > -1) &&
        (index(uc($v->get_name()), "FRIENDLY") > -1) ) {
my $half;
if (grep(/##/, $name)) {
$half = (split(/##/, $name))[1];
}
if (grep(/\?\\?/, $name)) {
$half = (split(/\?\\?/, $name))[1];
}
my ($dev, $sn) = (split(/#/, $half))[1,2];
$out_arr{$name}{value} = $dev." ".$sn." NAME:".$val;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$dev." ".$sn." ".$val."\t".$skey->get_timestamp());
}

elsif ( (index(uc($name), "SOFTWARE/MICROSOFT/INTERNET EXPLORER/TYPEDURLS") > -1) ) {
my $wseq = $v->get_name();
$wseq =~ s/url//;
$out_arr{$name}{value} = $wseq."~".$v->get_data();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."      ".$out_arr{$name}{value}."      ".$out_arr{$name}{time});
}

elsif ( (index(uc($name),
        "SOFTWARE/MICROSOFT/WINDOWS/CURRENTVERSION/APPLETS/WORDPAD/RECENT FILE LIST") > -1)
my $wseq = $v->get_name();
$wseq =~ s/File//;
$out_arr{$name}{value} = $wseq."~".$v->get_data();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message

```

```

# ::rptMsg($name."      ".$out_arr{$name}{value}."      ".$out_arr{$name}{time});
}
elseif ( (index(uc($name),"SOFTWARE/MICROSOFT/VCEXPRESS") > -1) &&
(index(uc($name),"FILEMRULIST") > -1) ) {
my $wseq = $v->get_name();
$wseq =~ s/File//;
$out_arr{$name}{value} = $wseq."~".$v->get_data();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."      ".$out_arr{$name}{value}."      ".$out_arr{$name}{time});
}
elseif ( (index(uc($name),"SOFTWARE/ADOBE/ACROBAT READER") > -1) &&
(index(uc($name),"AVGENERAL/CRECENTFILES") > -1) &&
(index(uc($v->get_name()),"TDITEXT") > -1) ) {
my $wseq = $skey->get_name();
$wseq =~ s/c//;
$out_arr{$name}{value} = $wseq."~".$v->get_data();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."      ".$out_arr{$name}{value}."      ".$out_arr{$name}{time});
}
elseif ( (index(uc($name),"SOFTWARE/MICROSOFT/OFFICE/14.0") > -1) &&
(index(uc($name),"FILE MRU") > -1) &&
(index(uc($v->get_name()),"ITEM") > -1) ) {
(my $prefix, my $suffix) = split(/\*/, $v->get_data());
$out_arr{$name}{value} = $suffix;
$out_arr{$name}{time} = getWinTS($v->get_data());
# Debug message
# ::rptMsg($name."      ".$out_arr{$name}{value}."      ".$out_arr{$name}{time});
}

elseif (
( (index(uc($name),"MRU") > -1 ) &&
(index(uc($name),"COMRUN") == -1) ) ||
( (index(uc($name),"TIME") > -1 ) &&
(index(uc($name),"QUICKTIME") == -1) &&
(index(uc($name),"RUNTIMEVERSION") == -1) &&
(index(uc($name),"COMPONENT") == -1) ) ||
( (index(uc($name),"DATE") > -1 ) &&
(index(uc($name),"UPDATE") == -1) &&
(index(uc($name),"VALIDATE") == -1) ) ||
(index(uc($name),"LASTFILES") > -1) ||
(index(uc($name),"TYPEDURL") > -1) ||
(index(uc($name),"RECENT") > -1 ) ) ) {

```

```

        if ( ($v->get_type() eq 1) || #REG_SZ
              ($v->get_type eq 2)) {    #REG_EXPAND_SZ
$out_arr{$name}{value} = $val;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$val."\t".gmtime($skey->get_timestamp()));
        } elsif ( ($v->get_type() eq 3) || #REG_BINARY
              ($v->get_type() eq 4)) { #REG_DWORD
$out_arr{$name}{value} = $v->get_data_as_string();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$v->get_data_as_string()."\t".gmtime($skey->get_timestamp()));
        } elsif ($v->get_type() eq 5) { # REG_BIG_ENDIAN
$out_arr{$name}{value} = $v->get_data_as_string();
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$v->get_data_as_string()."\t".gmtime($skey->get_timestamp()));
        }
    } elsif ($v->get_type() eq 1) { # REG_SZ
        if (isdateS($val)) {
$out_arr{$name}{value} = $val;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$val."\t".gmtime($skey->get_timestamp()));
        }
    } elsif ($v->get_type() eq 2) { #REG_EXPAND_SZ
        if (isdateS($val)) {
$out_arr{$name}{value} = $val;
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".$val."\t".gmtime($skey->get_timestamp()));
        }
    } elsif ($v->get_type() eq 3) { #REG_BINARY
    } elsif ($v->get_type() eq 4) { #REG_DWORD
        if (($val > $sdate) and ($val < $edate)) {
$out_arr{$name}{value} = gmtime($val);
$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".gmtime($val)."\t".gmtime($skey->get_timestamp()));
        }
    } elsif ($v->get_type() eq 5) { #REG_DWORD_BIG_ENDIAN
        if (($val > $sdate) and ($val < $edate)) {
$out_arr{$name}{value} = gmtime($val);

```

```

$out_arr{$name}{time} = $skey->get_timestamp();
# Debug message
# ::rptMsg($name."\t".gmtime($val)."\t".gmtime($skey->get_timestamp()));
    }
} elseif ($v->get_type() eq 6) { #REG_LINK
    } elseif ($v->get_type() eq 7) { #REG_MULTI_SZ
        if (scalar($val) > 0) {
            foreach my $subval ($val) {
                if (isdateS($subval)) {
                    $out_arr{$name}{value} = $subval;
                    $out_arr{$name}{time} = $skey->get_timestamp();
                    # Debug message
                    # ::rptMsg($name."\t".$subval."\t".gmtime($skey->get_timestamp()));
                }
            }
        }
    } elseif ($v->get_type() eq 8) { #RESOURCE_LIST
    } elseif ($v->get_type() eq 9) { #FLL_RSRC_DSCRPTR
    } elseif ($v->get_type() eq 10) { #RSRC_RQRMNT_LST
    } elseif ($v->get_type() eq 11) { #REG_QWORD
        if (($val > $sdate) and ($val < $edate)) {
            $out_arr{$name}{value} = gmtime($val);
            $out_arr{$name}{time} = $skey->get_timestamp();
            # Debug message
            # ::rptMsg($name."\t".gmtime($val)."\t".gmtime($skey->get_timestamp()));
        }
    }
}

#-----
# Check if the inputted string parameter is in one of
# Microsoft's string date formats
#-----
sub isdateS {
    my $wstring = shift;

    if (uc($wstring) =~ /\^\\d+\\/\\d+\\/\\d+$/ ) {
        return 1; # Any variation of 6/15/2009
    } elseif (uc($wstring) =~ /\^\\d+-\\d+-\\d+$/ ) {
        return 1; # Any variation of 6-15-2009
    } elseif (uc($wstring) =~ /\^\\d\\d?:\\d\\d (AM|PM)$/ ) {
        return 1; # 1:45 PM
    }
}

```

```

} elsif (uc($wstring) =~ /\d\d?:\d\d:\d\d (AM|PM)$/) {
    return 1; # 1:45:30 PM
} elsif (uc($wstring) =~ /\d+\/\d+\/\d+ \d\d?:\d\d (AM|PM)$/) {
    return 1; # 6/15/2009 1:45 PM
} elsif (uc($wstring) =~ /\d+\/\d+\/\d+ \d\d?:\d\d:\d\d (AM|PM)$/) {
    return 1; # 6/15/2009 1:45:30 PM
} elsif (uc($wstring) =~ /\d+-\d+-\d+ \d\d?:\d\d (AM|PM)$/) {
    return 1; # 6-15-2009 1:45 PM
} elsif (uc($wstring) =~ /\d+-\d+-\d+ \d\d?:\d\d:\d\d (AM|PM)$/) {
    return 1; # 6-15-2009 1:45:30 PM
} elsif (uc($wstring) =~ /^(1|2)\d\d\d-[01]\d-[0123]\d T \d\d?:\d\d:\d\d($| \d+$)/) {
    return 1; # 2009-06-15 T 13:45:30 possibly with 090000
} elsif (uc($wstring) =~ /^(1|2)\d\d\d-[01]\d-[0123]\d \d\d?:\d\d:\d\dZ$/) {
    return 1; # 2009-06-15 T 13:45:30Z
} elsif (uc($wstring) =~ /^(JANUARY|FEBRUARY|MARCH|APRIL|MAY|JUNE|JULY|AUGUST|SEPTEMBER|OCTOBER|NOVEMBER|DECEMBER)$/i) {
    return 1; # JUNE, 2009
} elsif (uc($wstring) =~ /^(JANUARY|FEBRUARY|MARCH|APRIL|MAY|JUNE|JULY|AUGUST|SEPTEMBER|OCTOBER|NOVEMBER|DECEMBER|JUNE 15)$/i) {
    return 1; # JUNE 15
} elsif (uc($wstring) =~ /^(MONDAY|TUESDAY|WEDNESDAY|THURSDAY|FRIDAY|SATURDAY|SUNDAY|MONDAY, JUNE 15)$/i) {
    return 1; # MONDAY, JUNE 15
} elsif (uc($wstring) =~ /^(MONDAY|TUESDAY|WEDNESDAY|THURSDAY|FRIDAY|SATURDAY|SUNDAY|MONDAY, JUNE 15, 2009)$/i) {
    return 1; # MONDAY, JUNE 15, 2009
} elsif (uc($wstring) =~ /^(MONDAY|TUESDAY|WEDNESDAY|THURSDAY|FRIDAY|SATURDAY|SUNDAY|MONDAY, 15 JUNE)$/i) {
    return 1; # MONDAY, 15 JUNE
}
return 0;
}

```

```

#-----
#
#-----
sub parsedate128 {
my $date = $_[0];
my @months = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
my @days = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
my ($yr, $mon, $dow, $dom, $hr, $min, $sec, $ms) = unpack("v*", $date);

$hr = "0".$hr if ($hr < 10);
$min = "0".$min if ($min < 10);
$sec = "0".$sec if ($sec < 10);
my $str = $days[$dow]. " ".$months[$mon - 1]. " ".$dom. " ".$hr. ":". $min. ":". $sec. " ".$yr;

```

```

return $str;
}

#-----
#
#-----
sub parsedate128_1 {
my $date = $_[0];
my ($yr,$mon,$dow,$dom,$hr,$min,$sec,$ms) = unpack("v*",$date);

$hr = "0".$hr if ($hr < 10);
$min = "0".$min if ($min < 10);
$sec = "0".$sec if ($sec < 10);
$mon = "0".$mon if ($mon < 10);
$dom = "0".$dom if ($dom < 10);
my $str = $mon."/".$dom."/".$yr." ".$hr.".".$min.".".$sec." (dow:".$dow.)";

return $str;
}

#-----
#
#-----
sub parseShellItem {
my $data = shift;
my $len = length($data);
my $str;

my $tag = 1;
my $cnt = 0;
while ($tag) {
my %item = ();
my $sz = unpack("v",substr($data,$cnt,2));
$tag = 0 if (($sz == 0) || ($cnt + $sz > $len));

my $dat = substr($data,$cnt,$sz);
my $type = unpack("C",substr($dat,2,1));

if ($type == 0x1F) { #System Folder
$item = parseSystemFolderEntry($dat);
$str .= "\\\".$item{name};
} elsif ($type == 0x2F) { #Volume (drive letter)
$item = parseDriveEntry($dat);

```



```

$item{name} =~ s/\\$//;
$str .= "\\\".$item{name};
} elsif ( ($type == 0x31) || ($type == 0x32) || ($type == 0x3a) || ($type == 0x74)) {
$item = parseFolderEntry($dat);
$str .= "\\\".$item{name};
} elsif ($type == 0x00) {
} elsif ( ($type == 0xc3) || ($type == 0x41) || ($type == 0x42) || ($type == 0x46) ||
my $id = unpack("C",substr($dat,3,1));
if ( ($type == 0xc3) && ($id != 0x01)) {
$item = parseNetworkEntry($dat);
} else {
$item = parseNetworkEntry($dat);
}
$str .= "\\\".$item{name};
}
else {
$item{name} = sprintf "Unknown Type (0x%x)", $type;
$str .= "\\\".$item{name};
}
$cnt += $sz;
}
$str =~ s/^\\$//;
return $str;
}

```

```

#-----
#
#-----
sub parseSystemFolderEntry {
my $data = shift;
my %item = ();

my %vals = (0x00 => "Explorer",
            0x42 => "Libraries",
            0x44 => "Users",
            0x4c => "Public",
            0x48 => "My Documents",
            0x50 => "My Computer",
            0x58 => "My Network Places",
            0x60 => "Recycle Bin",
            0x68 => "Explorer",
            0x70 => "Control Panel",
            0x78 => "Recycle Bin",

```

```

0x80 => "My Games");

$item{type} = unpack("C",substr($data,2,1));
$item{id}   = unpack("C",substr($data,3,1));
if (exists $vals{$item{id}}) {
$item{name} = $vals{$item{id}};
}
else {
$item{name} = parseGUID(substr($data,4,16));
}
return %item;
}

#-----
# parseGUID()
# Takes 16 bytes of binary data, returns a string formatted
# as an MS GUID.
#-----
sub parseGUID {
my $data      = shift;
my $d1 = unpack("V",substr($data,0,4));
my $d2 = unpack("v",substr($data,4,2));
my $d3 = unpack("v",substr($data,6,2));
my $d4 = unpack("H*",substr($data,8,2));
my $d5 = unpack("H*",substr($data,10,6));
return sprintf "{%08x-%x-%x-$d4-$d5}",$d1,$d2,$d3;
}

#-----
#
#-----
sub parseDriveEntry {
my $data      = shift;
my %item = ();
$item{type} = unpack("C",substr($data,2,1));
$item{name} = substr($data,3,3);
return %item;
}

#-----
# parseNetworkEntry()
#
#-----
sub parseNetworkEntry {

```

```

my $data = shift;
my %item = ();
$item{type} = unpack("C",substr($data,2,1));

my @n = split(/\00/,substr($data,4,length($data) - 4));
$item{name} = $n[0];
$item{name} =~ s/^\W//;
return %item;
}
#-----
#
#-----

sub parseFolderEntry {
my $data      = shift;
my %item = ();

$item{type} = unpack("C",substr($data,2,1));
# Type 0x74 folders have a slightly different format

my $ofs_mdate;
my $ofs_shortname;

if ($item{type} == 0x74) {
$ofs_mdate = 0x12;
}
elsif (substr($data,4,4) eq "AugM") {
$ofs_mdate = 0x1c;
}
elsif ($item{type} == 0x31 || $item{type} == 0x32 || $item{type} == 0x3a) {
$ofs_mdate = 0x08;
}
else {}
# some type 0x32 items will include a file size
if ($item{type} == 0x32) {
my $size = unpack("V",substr($data,4,4));
if ($size != 0) {
$item{filesize} = $size;
}
}

my @m = unpack("vv",substr($data,$ofs_mdate,4));
($item{mtime_str},$item{mtime}) = convertDOSDate($m[0],$m[1]);

```

```

# Need to read in short name; nul-term ASCII
# $item{shortname} = (split(/\00/,substr($data,12,length($data) - 12),2))[0];
$ofs_shortname = $ofs_mdate + 6;
my $tag = 1;
my $cnt = 0;
my $str = "";
while($tag) {
my $s = substr($data,$ofs_shortname + $cnt,1);
if ($s =~ m/\00/ && ((( $cnt + 1) % 2) == 0)) {
$tag = 0;
}
else {
$str .= $s;
$cnt++;
}
}
# $str =~ s/\00//g;
my $shortname = $str;
my $ofs = $ofs_shortname + $cnt + 1;
# Read progressively, 1 byte at a time, looking for Oxbeef
my $tag = 1;
my $cnt = 0;
while ($tag) {
if (unpack("v",substr($data,$ofs + $cnt,2)) == 0xbeef) {
$tag = 0;
}
else {
$cnt++;
}
}
$item{extver} = unpack("v",substr($data,$ofs + $cnt - 4,2));

# ::rptMsg(sprintf " BEEF Offset: 0x%x", $ofs + $cnt);
# ::rptMsg(" Version: ".$item{extver});

$ofs = $ofs + $cnt + 2;

my @m = unpack("vv",substr($data,$ofs,4));
($item{ctime_str},$item{ctime}) = convertDOSDate($m[0],$m[1]);
$ofs += 4;
my @m = unpack("vv",substr($data,$ofs,4));
($item{atime_str},$item{atime}) = convertDOSDate($m[0],$m[1]);
$ofs += 4;

```

```

my $jmp;
if ($item{extver} == 0x03) {
    $jmp = 8;
}
elseif ($item{extver} == 0x07) {
    $jmp = 22;
}
elseif ($item{extver} == 0x08) {
    $jmp = 26;
}
else {}

$ofs += $jmp;
# ::rptMsg(sprintf "   Offset: 0x%x", $ofs);

my $str = substr($data, $ofs, length($data) - $ofs);

my $longname = (split(/\00\00/, $str, 2))[0];
$longname =~ s/\00//g;

if ($longname ne "") {
    $item{name} = $longname;
}
else {
    $item{name} = $shortname;
}
return %item;
}

#-----
# parseVariableEntry()
#
#-----
sub parseVariableEntry {
    my $data      = shift;
    my %item = ();

    $item{type} = unpack("C", substr($data, 2, 1));
    my $tag = unpack("C", substr($data, 0x0A, 1));

    if (unpack("v", substr($data, 4, 2)) == 0x1A) {
        my $guid = parseGUID(substr($data, 14, 16));
    }
}

```

```

if (exists $folder_types{$guid}) {
$item{name} = $folder_types{$guid};
}
else {
$item{name} = $guid;
}
}
elsif (grep(/1SPS/, $data)) {
my @seg = split(/1SPS/, $data);

my %segs = ();
foreach my $s (0..(scalar(@seg) - 1)) {
my $guid = parseGUID(substr($seg[$s], 0, 16));
$segs{$guid} = $seg[$s];
}

if (exists $segs{"{b725f130-47ef-101a-a5f1-02608c9eebac}")) {
# Ref: http://msdn.microsoft.com/en-us/library/aa965725\(v=vs.85\).aspx
my $stuff = $segs{"{b725f130-47ef-101a-a5f1-02608c9eebac}"};

my $tag = 1;
my $cnt = 0x10;
while($tag) {
my $sz = unpack("V", substr($stuff, $cnt, 4));
my $id = unpack("V", substr($stuff, $cnt + 4, 4));
#-----
# sub-segment types
# 0x0a - file name
# 0x14 - short name
# 0x0e, 0x0f, 0x10 - mod date, create date, access date(?)
# 0x0c - size
#-----
if ($sz == 0x00) {
$tag = 0;
next;
}
elsif ($id == 0x0a) {

my $num = unpack("V", substr($stuff, $cnt + 13, 4));
my $str = substr($stuff, $cnt + 13 + 4, ($num * 2));
$str =~ s/\\00//g;
$item{name} = $str;

```

```

    }
    $cnt += $sz;
  }
}

# if (exists $segs{"{5cbf2787-48cf-4208-b90e-ee5e5d420294}"} ) {
# my $stuff = $segs{"{5cbf2787-48cf-4208-b90e-ee5e5d420294}"};
#   my $tag = 1;
#   my $cnt = 0x10;
#   while($tag) {
#     my $sz = unpack("V",substr($stuff,$cnt,4));
#     my $id = unpack("V",substr($stuff,$cnt + 4,4));
#
#     if ($sz == 0x00) {
#       $tag = 0;
#       next;
#     }
#     elsif ($id == 0x19) {
#
#       my $num = unpack("V",substr($stuff,$cnt + 13,4));
#       my $str = substr($stuff,$cnt + 13 + 4,($num * 2));
#       $str =~ s/\00//g;
#       $item{name} = $str;
#     }
#     $cnt += $sz;
#   }
# }
}

elsif (substr($data,4,4) eq "AugM") {
%item = parseFolderEntry($data);
}

# Following two entries are for Device Property data
elsif ($tag == 0x7b || $tag == 0xbb || $tag == 0xfb) {
my ($sz1,$sz2,$sz3) = unpack("VVV",substr($data,0x3e,12));
$item{name} = substr($data,0x4a,$sz1 * 2);
$item{name} =~ s/\00//g;
}

elsif ($tag == 0x02 || $tag == 0x03) {
my ($sz1,$sz2,$sz3,$sz4) = unpack("VVVV",substr($data,0x26,16));
$item{name} = substr($data,0x36,$sz1 * 2);
$item{name} =~ s/\00//g;
}

else {

```

```

$item{name} = "Unknown Type";
}
return %item;
}

#-----
# parseNetworkEntry2()
#
#-----
sub parseNetworkEntry2 {
my $data = shift;
my %item = ();
$item{type} = unpack("C",substr($data,2,1));

my @n = split(/\00/,substr($data,4,length($data) - 4));
$item{name} = $n[0];
return %item;
}

#-----
# parseZipSubFolderItem()
# parses what appears to be Zip file subfolders; this type
# appears to contain the date and time of when the subfolder
# was accessed/opened, in string format.
#-----
sub parseZipSubFolderItem {
my $data = shift;
my %item = ();

# Get the opened/accessed date/time
$item{datetime} = substr($data,0x24,6);
$item{datetime} =~ s/\00//g;
if ($item{datetime} eq "N/A") {

}
else {
$item{datetime} = substr($data,0x24,40);
$item{datetime} =~ s/\00//g;
my ($date,$time) = split(/\s+/, $item{datetime},2);
my ($mon,$day,$yr) = split(/\//,$date,3);
my ($hr,$min,$sec) = split(/:/$, $time,3);
my $gmttime = timegm($sec,$min,$hr,$day,($mon - 1),$yr);
# $item{datetime} = "$yr-$mon-$day $hr:$min:$sec";

```



```

$item{datetime} = $mon."/".$day."/".$yr." ".$hr.":".$min.":".$sec." (dow: 00)";
# ::rptMsg("[Access_Time]: ".gmtime($gmtime));
}

my $sz = unpack("V",substr($data,0x54,4));
my $sz2 = unpack("V",substr($data,0x58,4));

my $str1 = substr($data,0x5C,$sz * 2) if ($sz > 0);
$str1 =~ s/\00//g;
my $str2 = substr($data,0x5C + ($sz * 2),$sz2 * 2) if ($sz2 > 0);
$str2 =~ s/\00//g;

if ($sz2 > 0) {
# $item{name} = $str1."\".$str2;
$item{name} = $str1."/".$str2; # JSO 29-Jan-14, changed from backslash to forward slash
}
else {
$item{name} = $str1;
}
return %item;
}

#-----
# parse01ShellItem()
# I honestly have no idea what to do with this data; there's really
# no reference for or description of the format of this data. For
# now, this is just a place holder
#-----
sub parse01ShellItem {
my $data = shift;
my %item = ();
$item{type} = unpack("C",substr($data,2,1));;
$item{name} = "";
# ($item{val0},$item{val1}) = unpack("VV",substr($data,2,length($data) - 2));
return %item;
}

#-----
#
#-----
sub parseURIEntry {
my $data = shift;
my %item = ();

```

```

$item{type} = unpack("C",substr($data,2,1));

my ($lo,$hi) = unpack("VV",substr($data,0x0e,8));
$item{uritime} = ::getTime($lo,$hi);

my $sz = unpack("V",substr($data,0x2a,4));
my $uri = substr($data,0x2e,$sz);
$uri =~ s/\00//g;

my $proto = substr($data,length($data) - 6, 6);
$proto =~ s/\00//g;

$item{name} = $proto."://".$uri." [" .gmtime($item{uritime})."]";

return %item;
}

#-----
#
#-----
sub parseDeviceEntry {
my $data = shift;
my %item = ();

my $userlen = unpack("V",substr($data,30,4));
my $devlen  = unpack("V",substr($data,34,4));

my $user     = substr($data,0x28,$userlen * 2);
$user =~ s/\00//g;

my $dev = substr($data,0x28 + ($userlen * 2),$devlen * 2);
$dev =~ s/\00//g;

$item{name} = $user;
return %item;
}

#-----
#
#-----
sub parseControlPanelEntry {
my $data     = shift;
my %item = ();

```

```

$item{type} = unpack("C",substr($data,2,1));
my $guid = parseGUID(substr($data,14,16));
if (exists $cp_guids{$guid}) {
$item{name} = $cp_guids{$guid};
}
else {
$item{name} = $guid;
}
return %item;
}

#-----
# parseFolderEntry2()
#
# Initial code for parsing type 0x35
#-----

sub parseFolderEntry2 {
my $data      = shift;
my %item = ();

my $ofs = 0;
my $tag = 1;

while ($tag) {
if (unpack("v",substr($data,$ofs,2)) == 0xbeef) {
$tag = 0;
}
else {
$ofs++;
}
}
$item{extver} = unpack("v",substr($data,$ofs - 4,2));
# Move offset over to end of where the ctime value would be
$ofs += 4;

my $jmp;
if ($item{extver} == 0x03) {
$jmp = 8;
}
elsif ($item{extver} == 0x07) {
$jmp = 26;
}
elsif ($item{extver} == 0x08) {

```

```

$jmp = 30;
}
else {}

$ofs += $jmp;

my $str = substr($data,$ofs,length($data) - 30);

#::rptMsg(" --- parseFolderEntry2 --- ");
my @d = printData($str);
foreach (0..(scalar(@d) - 1)) {
# ::rptMsg($d[$_]);
}
#::rptMsg("");

$item{name} = (split(/\00\00/,$str,2))[0];
$item{name} =~ s/\13\20/\2D\00/;
$item{name} =~ s/\00//g;

return %item;
}
#-----
#
#-----
sub parseNetworkEntry {
my $data = shift;
my %item = ();
$item{type} = unpack("C",substr($data,2,1));
my @names = split(/\00/,substr($data,5,length($data) - 5));
$item{name} = $names[0];
return %item;
}

#-----
# convertDOSDate()
# subroutine to convert 4 bytes of binary data into a human-
# readable format. Returns both a string and a Unix-epoch
# time.
#-----
sub convertDOSDate {

```

```

my $date = shift;
my $time = shift;

if ($date == 0x00 || $time == 0x00){
return (0,0);
}
else {
my $sec = ($time & 0x1f) * 2;
$sec = "0".$sec if (length($sec) == 1);
if ($sec == 60) {$sec = 59};
my $min = ($time & 0x7e0) >> 5;
$min = "0".$min if (length($min) == 1);
my $hr = ($time & 0xF800) >> 11;
$hr = "0".$hr if (length($hr) == 1);
my $day = ($date & 0x1f);
$day = "0".$day if (length($day) == 1);
my $mon = ($date & 0x1e0) >> 5;
$mon = "0".$mon if (length($mon) == 1);
my $yr = (($date & 0xfe00) >> 9) + 1980;
my $gmtime = timegm($sec,$min,$hr,$day,($mon - 1),$yr);

my $str = $mon."/". $day."/". $yr." ".$hr.":". $min.":". $sec." (dow: 00)";
return $str;

# return ("Yr-$mon-$day $hr:$min:$sec",$gmtime);
# return gmtime(timegm($sec,$min,$hr,$day,($mon - 1),$yr));
}
}

sub getWinTS {
my $data = shift;
my $winTS;
my $dateTime;
(my $prefix, my $suffix) = split(/\*/, $data);
if ($prefix =~ /\[.{9}\]\[T(.{16})\]/) {
$winTS = $1;
my @vals = split(//, $winTS);
my $t0 = join('', @vals[0..7]);
my $t1 = join('', @vals[8..15]);
$dateTime = ::getTime(hex($t1), hex($t0));
}
return ($dateTime);
}

```

1;

Appendix F. Terms and Strings

ActivityBag

Print,Search

ConsolidateStrings,CreateRules ItemUtilities,Utilities,TearCPP,Parameters,Timeline

6.1 Generics

Header.

```
#pragma once
#include <set>
#include <unordered_map>
#include "TearCPP.h"
#include "Parameters.h"
#define htItemGenerics_i std::unordered_map<uint32_t, uint32_t>::const_iterator
#define htGenericItems_i std::unordered_map<uint32_t, genericItem_t>::const_iterator

class Generics {

private:
    typedef struct genericItem_type {
        uint32_t genericItem;
        uint32_t itemid;
        uint8_t inDB;
    } genericItem_t;

    sqlite3_stmt *stmt_selGenericItem;
    sqlite3_stmt *stmt_insGenericItems;
    sqlite3_stmt *stmt_insItemGenerics;

    Parameters *parameters;
    uint32_t maxGeneric;
    std::unordered_map<uint32_t, genericItem_t> htGenericItems;
    std::unordered_map<uint32_t, uint32_t> htItemGenerics;

public:
    Generics(Parameters *wparameters);
    ~Generics();
    uint8_t isGenericItem(uint32_t witemid);
    uint32_t getGeneric(uint32_t witemid);
    uint32_t getItem(uint32_t wgeneric);
    void loadGenerics();
    void writeGenericItems();
```

```

        uint32_t addGenericItem();

};

```

C++.

```

//
// Class: Generics
// Notes:
//      1. Generics represents the variables of this project. They are assigned//
//      to terms so that multiple items can be inserted into the terms. They//
//      are assigned to patterns to restrict the patterns of the item names //
//      that fit into a specific term.
// Initial Release: 29-Sep-17
//
// Revision History:
//
////////////////////////////////////
#pragma once
#include "Generics.h"

Generics::Generics(Parameters *wparameters) {
    char exec_string[2000];
    int rc;

    parameters = wparameters;

    maxGeneric = 0;
    sprintf_s(exec_string, 2000, "SELECT genericItem, itemID FROM genericItems;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selGenericItem, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare Select Generic Item Statement\n", rc);
        return;
    }

    sprintf_s(exec_string, 2000, "insert INTO items (itemid, itemtypeid, tracetype, tr");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insItemGenerics, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare Insert Item Generics Statement\n", rc);
        return;
    }

    sprintf_s(exec_string, 2000, "insert INTO genericItems (genericItem, itemID) VALUE

```



```

        rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insGenericItems,
        if (rc != SQLITE_OK) {
            printf("Error: %i. couldn't prepare Insert Generic Items Statement\n", rc);
            return;
        }
    }

}

Generics::~Generics() {}

uint8_t Generics::isGenericItem(uint32_t witemid) {
    if (htItemGenerics.find(witemid) != htItemGenerics.end())
        return 1;
    else
        return 0;
}

uint32_t Generics::getGeneric(uint32_t witemid) {
    htItemGenerics_i htiter = htItemGenerics.find(witemid);
    if (htiter != htItemGenerics.end()) return (htiter->second);
    return 0;
}

uint32_t Generics::getItem(uint32_t wgeneric) {
    htGenericItems_i htiter = htGenericItems.find(wgeneric);
    if (htiter != htGenericItems.end()) return (htiter->second.itemid);
    return 0;
}

void Generics::loadGenerics() {
    htGenericItems_i htiter;

    int rc;
    uint32_t witemid, wgeneric;
    genericItem_t witem2;

    maxGeneric = 0;
    sqlite3_reset(stmt_selGenericItem);
    if (parameters->debugParms >= 6) fprintf(parameters->logFile, "Loading Generics\n");
    while ((rc = sqlite3_step(stmt_selGenericItem)) == SQLITE_ROW) {
        wgeneric = sqlite3_column_int(stmt_selGenericItem, 0);
        if (wgeneric > maxGeneric) maxGeneric = wgeneric;
    }
}

```

```

        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "\tgeneric item %i      max = %i\n",
                    wgeneric, maxGeneric);

        witemid = sqlite3_column_int(stmt_selGenericItem, 1);
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "\t\titemid %i\n", witemid);

        witem2.genericItem = wgeneric;
        witem2.itemid = witemid;
        witem2.inDB = 1;
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "\t\tcontainer created\n");

        htGenericItems[wgeneric] = witem2;
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "\t\tadded to hash table\n");
        htItemGenerics[witemid] = wgeneric;
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "\t\tadded to hash table\n");
    }
}

void Generics::writeGenericItems() {
    htGenericItems_i htiter;

    genericItem_t witem;
    int rc;

    char genericName[52];
    for (htiter = htGenericItems.begin(); htiter != htGenericItems.end(); htiter++) {
        witem = (htiter->second);
        if (witem.inDB == 0) {
            sprintf_s(genericName, 50, "Generic Item %u", witem.genericItem);
            sqlite3_reset(stmt_insItemGenerics);
            sqlite3_bind_int(stmt_insItemGenerics, 1, witem.itemid);
            sqlite3_bind_text(stmt_insItemGenerics, 2, genericName, -1,
                             SQLITE_TRANSIENT);
            rc = sqlite3_step(stmt_insItemGenerics);
            sqlite3_reset(stmt_insGenericItems);
            sqlite3_bind_int(stmt_insGenericItems, 1, witem.genericItem);
            sqlite3_bind_int(stmt_insGenericItems, 2, witem.itemid);
            rc = sqlite3_step(stmt_insGenericItems);

```

```

    }
}

```

6.2 GenericsXref

Header.

```

#pragma once
#include "TearCPP.h"
#include "lists.h"
#include "ItemInstance.h"

class GenericXref {
    uint32_t itemid;
    uint32_t genericType;
    ItemInstance *itemList;
    listT<ItemInstance> *loopList;
    uint32_t genericItem;
    uint32_t listid;

public:
    GenericXref(uint32_t wgeneric, uint32_t witemid, uint32_t wgenericType,
        ItemInstance *witemList, listT<ItemInstance> *wloopList, uint32_t wlistid) :
        itemid(witemid), genericItem(wgeneric), genericType(wgenericType),
        itemList(witemList != NULL ? new ItemInstance(witemList) : NULL),
        loopList(wloopList != NULL ? wloopList->copyClass() : NULL),
        listid(wlistid) {}

    GenericXref() :
        itemid(0), genericItem(0), genericType(GNRC_ITEM),
        itemList(NULL), loopList(NULL),
        listid(0) {}

    GenericXref(uint32_t wgeneric, uint32_t witemid) :
        itemid(witemid), genericItem(wgeneric), genericType(GNRC_ITEM),
        itemList(NULL), loopList(NULL),
        listid(0) {}

    GenericXref(uint32_t wgeneric, ItemInstance *witemList) :
        itemid(0), genericItem(wgeneric), genericType(GNRC_ITEMLIST),
        itemList(witemList != NULL ? new ItemInstance(witemList) : NULL), loopList(NULL),
        listid(0) {}

    GenericXref(uint32_t wgeneric, listT<ItemInstance> *wloopList, uint32_t wlistid) :

```

```

    itemid(0), genericItem(wgeneric), genericType(GNRC_ITEMLIST),
    itemList(NULL), loopList(wloopList != NULL ? wloopList->copyClass() : NULL),
    listid(wlistid) {}

GenericXref(GenericXref *other) {
    if (other == NULL) {
        itemid = 0;
        genericItem = 0;
        genericType = GNRC_ITEM;
        itemList = NULL;
        loopList = NULL;
        listid = 0;
    }
    else {
        itemid = other->getItem();
        genericItem = other->getGeneric();
        genericType = other->getGenericType();
        if (other->getItemList() != NULL)
            itemList = (other->getItemList())->itemInstanceCopy();
        else
            itemList = NULL;
        if (other->getLoopList() != NULL)
            loopList = (other->getLoopList())->copyClass();
        else
            loopList = NULL;
        listid = other->getListid();
    }
}

~GenericXref() {
    void *liter;
    ItemInstance *witem;

    if (itemList != NULL) delete itemList;
    if (loopList != NULL) {
        if (!loopList->lempty()) {
            for (liter = loopList->getFront(); liter != NULL;
                 liter = loopList->revIterate(liter)) {
                witem = loopList->lvalue(liter);
                delete witem;
            }
        }
        delete loopList;
    }
}

```

```

    }
}

uint8_t setGeneric(uint32_t wgenericItem) { genericItem = wgenericItem; return 1; }
uint8_t setGenericType(uint32_t wgenericType) { genericType = wgenericType; return 1; }
uint8_t setItem(uint32_t witemid) { itemid = witemid; return 1; }
uint8_t setListid(uint32_t wlistid) { listid = wlistid; return 1; }
uint8_t setItemList(ItemInstance *witemList) {
    if (witemList != NULL)
        itemList = witemList->itemInstanceCopy();
    else
        itemList = NULL;
    return 1;
}
uint8_t setLoopList(listT<ItemInstance> *wloopList) {
    if (wloopList != NULL)
        loopList = wloopList->copyClass();
    else
        loopList = NULL;
    return 1;
}
uint32_t getGeneric() { return genericItem; }
uint32_t getItem() { return itemid; }
uint32_t getListid() { return listid; }
uint32_t getGenericType() { return genericType; }
ItemInstance *getItemList() { return itemList; }
listT<ItemInstance> *getLoopList() { return loopList; }
};

```

6.3 InstanceGenericsXref

Header.

```

#pragma once
#include "TearCPP.h"

class InstanceGenericXref
{
    char *order;
    uint32_t oldGeneric;
    uint32_t newGeneric;

public:
    InstanceGenericXref() : order(NULL), oldGeneric(0), newGeneric(0) {}

```

```

~InstanceGenericXref() { if (order != NULL) free(order); }

InstanceGenericXref(char *worder, uint32_t woldGeneric, uint32_t wnewGeneric) {
    if (worder != NULL) {
        order = (char *)malloc(sizeof(char) * (strlen(worder) + 2));
        strcpy_s(order, strlen(worder) + 1, worder);
    }
    else order = NULL;

    oldGeneric = woldGeneric;
    newGeneric = wnewGeneric;
}

InstanceGenericXref(InstanceGenericXref *other) {
    if (other->order != NULL) {
        order = (char *)malloc(sizeof(char) * (strlen(other->order) + 2));
        strcpy_s(order, strlen(other->order) + 1, other->order);
    }
    else order = NULL;

    oldGeneric = other->oldGeneric;
    newGeneric = other->newGeneric;
}

char *getOrder() { return order; }
uint32_t getOldGeneric() { return oldGeneric; }
uint32_t getNewGeneric() { return newGeneric; }

};

```

6.4 Patterns

Header.

```
#pragma once
#include <unordered_map>
#include "Generics.h"
#include "queues.h"
#include "lists.h"
#include "utilities.h"
#include "ItemUtilities.h"
#include "ItemInstance.h"
#include "TearCPP.h"
#define htItems_i std::unordered_map<uint32_t, Utilities::item_t *>::const_iterator
#define htItemPatterns_i std::unordered_map<uint32_t, listT<genericPatterns>>::const_iterator
#define htWordList_i std::unordered_map<std::string, wordBox *>::const_iterator

class Generics;
class Parameters;
class ItemUtilities;

class Patterns {

public:
    typedef struct genericPatternType {
        listT<unsigned int> *itemList;
        char *pattern;
        uint8_t inDB;
        uint32_t id;
        struct genericPatternType *next;
    } genericPatterns;

private:

    std::unordered_map<uint32_t, uint32_t> htItemPatterns;
    listT<genericPatterns> **itemPatterns;

    typedef struct itemBox_t {
        uint32_t itemid;
        uint32_t itemtypeid;
        std::string *name;
        uint32_t *words;
        uint32_t wordCount;
        uint32_t group;
```

```

} itemBox;

typedef struct wordBox_t {
    std::string *name;
    uint32_t group;
    uint32_t length;
    uint32_t id;
    struct wordBox_t *prev;
    struct wordBox_t *next;
    uint32_t count;
} wordBox;

typedef struct phraseBox_t {
    uint32_t *id;
    uint32_t length;
    struct phraseBox_t *next;
} phraseBox;

typedef struct phrase_t {
    uint32_t *ids;
    uint32_t length;
    uint32_t wordCount;
    uint32_t count;
} phraseType;

genericPatterns *patternHead;
genericPatterns *starPattern;
Utilities *utilities;
Parameters *parameters;
Generics *generics;
ItemUtilities *itemUtilities;

sqlite3_stmt *stmt_selItems;
sqlite3_stmt *stmt_selItemPatternCount;
sqlite3_stmt *stmt_selItemPatterns;
sqlite3_stmt *stmt_insItemPatterns;
sqlite3_stmt *stmt_selPattern;
sqlite3_stmt *stmt_selPatternGenerics;
sqlite3_stmt *stmt_selPatternGenerics2;
sqlite3_stmt *stmt_insPatterns;
sqlite3_stmt *stmt_insPatternGenerics;

void getWords(std::list<std::string>*qNames, std::string itemName);

```



```

void getSeparator(char *wstring, char **separatorStart, char **separatorEnd);
uint32_t Patterns::numberCount(std::string wname);
uint32_t insertWord(std::unordered_map<std::string, wordBox *> *htWordList,
    std::string wname, uint32_t *wordCount, uint32_t group);
uint32_t Patterns::insertPhrase(wordBox **head, std::string wname,
    uint32_t *windex);
std::string getPhrase(wordBox language[], uint32_t wordCount, uint32_t *ids);
void insertPattern(genericPatterns *newPattern);
uint32_t getGroup(char *name, uint32_t itemtype);
double getPerc(uint32_t wgroup);

public:
    Patterns(Utilities *utilities, ItemUtilities *witemUtilities,
        Parameters *parameters, Generics *generics);
    ~Patterns();
    void loadPatterns();
    void createPatterns();
    uint32_t getPattern(uint32_t witemid);
    uint32_t getGenericMatch(char *wpattern, listT<uint32_t> *usedItems);
    uint32_t getGenericMatch(char *wname, listT<uint32_t> **usedItems,
        uint8_t allowAdd);
    uint32_t getGenericMatch(uint32_t witemid, listT<uint32_t> **usedItems,
        uint8_t allowAdd);
    void writePatterns();
    genericPatterns *getPatternHead() { return patternHead; }
    genericPatterns *getStarPattern() { return starPattern; }
};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Patterns                                                                                   //
// Notes:                                                                                           //
//     1. Patterns restrict the scope of the associated generics when they //
//        are placed in terms. //
//     2. For instance a generic item with a pattern of ^.:/Windows/System/.*$ //
//        means that terms with this generic in them will only match files whose //
//        names match that pattern. //
//     3. This class also includes an automated way for creating these patterns. //
//        It looks at all of the item names and creates all possible patterns //
//        from them. It then keeps those patterns that occur in more than X% of //
//        the items. X is user defined and requires changing this class. The //
//        routine is divided into subsets of items because handling all of them //

```

```

//          at once overwhelms the memory and takes too long.          //
// Initial Release: 29-Sep-17                                          //
//                                                                    //
// TODO: 1. Enhance CreatePatterns to allow the different subsets and thresholds //
//          to be passed via the parameters file.                      //
//                                                                    //
// Revision History:                                                  //
//                                                                    //
////////////////////////////////////
#include "Patterns.h"

Patterns::Patterns(Utilities *wutilities, ItemUtilities *witemUtilities,
                  Parameters *wparameters, Generics *wgenerics) {
    int rc;
    char exec_string[2000];
    sqlite3_stmt *stmt;

    patternHead = NULL;
    starPattern = NULL;
    itemUtilities = witemUtilities;
    utilities = wutilities;
    parameters = wparameters;
    generics = wgenerics;

    // These statements will fail after the first time, but not hurt anything
    sprintf_s(exec_string, 2000,
              "Create TABLE itemPatterns(ItemID INTEGER, PatternID INT4);");
    sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    sqlite3_step(stmt);
    sqlite3_finalize(stmt);
    sprintf_s(exec_string, 2000,
              "Create INDEX itemPs ON itemPatterns (itemid,patternID);");
    sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    sqlite3_step(stmt);
    sqlite3_finalize(stmt);

    // Using item patterns incurs a high up front time penalty but significantly
    // speeds processing.
    if (parameters->useItemPatternLists) {
        itemPatterns = (listT<genericPatterns>**)
            malloc(sizeof(listT<genericPatterns>*) * (parameters->maxItemid + 2));
        for (int i = 0; i <= parameters->maxItemid; i++) itemPatterns[i] = NULL;
    }
}

```

```

} else
    itemPatterns = NULL;

// SQL Statements
sprintf_s(exec_string, 2000, "SELECT count(*) FROM itemPatterns;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000,
                        &stmt_selItemPatternCount, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Item Pattern Count Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT patternId FROM itemPatterns WHERE itemId = ?1");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selItemPatterns,
                        &stmt_selItemPatternCount, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Item Patterns Statement\n", rc);
}

sprintf_s(exec_string, 2000,
          "INSERT INTO itemPatterns (itemId, patternId) VALUES (?1, ?2);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insItemPatterns,
                        &stmt_selItemPatternCount, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Item Patterns Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT itemId FROM items;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selItems, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Items Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT id, pattern FROM patterns;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selPattern, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Patterns Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT itemId FROM patternGenerics WHERE id = ?1;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selPatternGeneric,
                        &stmt_selItemPatternCount, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Pattern Generics Statement\n", rc);
}

sprintf_s(exec_string, 2000,

```

```

        "SELECT itemid FROM patternGenerics WHERE id = ?1 AND itemid = ?2;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selPatternGeneric
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Pattern Generics Statement\n", rc);
}

sprintf_s(exec_string, 2000, "insert INTO patterns (pattern) VALUES(?1);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insPatterns, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Patterns Statement\n", rc);
}

sprintf_s(exec_string, 2000, "insert INTO patternGenerics (id, itemid) VALUES(?1,
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insPatternGeneric
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert PatternGenerics Statement\n", rc);
}
}

Patterns::~Patterns() {}

uint32_t Patterns::getPattern(uint32_t witemid) {
    std::unordered_map<uint32_t, uint32_t>::const_iterator htIter;

    htIter = htItemPatterns.find(witemid);
    if (htIter != htItemPatterns.end()) return htIter->second;
    return 0;
}

void Patterns::loadPatterns() {
    // Purpose: Loads the patterns from the DB into an unordered map for faster
    // processing
    htItem_i htiter;
    ItemUtilities::item_t *itemDef;

    int rc, rc2;
    uint32_t ctr, last_seen;
    uint32_t *wid, *witemid;
    char *spattern;
    genericPatterns *wpattern, *wtail;
    listT<genericPatterns> *wpatternList;

```



```

    }

    // Load the generic items for this pattern
    sqlite3_reset(stmt_selPatternGenerics);
    sqlite3_bind_int(stmt_selPatternGenerics, 1, *wid);
    while ((rc2 = sqlite3_step(stmt_selPatternGenerics)) == SQLITE_ROW) {
        witemid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
        *witemid = sqlite3_column_int(stmt_selPatternGenerics, 0);
        if (*witemid > 0)
            wpattern->itemList->insertRear(witemid);
        else
            free(witemid);
        htItemPatterns[*witemid] = *wid;
    }
}

// Do this once -- figure out what patterns match which items
//      -- large up front cost by linking all individual items with
//      generic items (and patterns) before doing any processing,
//      but still significantly speeds subsequent processing
if (parameters->useItemPatternLists) {

    // Check if we've already done this; if so, just use what we did before
    sqlite3_reset(stmt_selItemPatternCount);
    sqlite3_step(stmt_selItemPatternCount);
    printf("Linking %i items to patterns\n", parameters->maxItemid);
    if (sqlite3_column_int(stmt_selItemPatternCount, 0) > 0) {
        printf("\tloading existing item/patterns\n");
        uint32_t maxPattern = 0;
        for (wpattern = patternHead; wpattern != NULL; wpattern = wpattern->next) {
            if (wpattern->id > maxPattern) maxPattern = wpattern->id;
        }
        genericPatterns **patternXref = (genericPatterns **)
            malloc(sizeof(genericPatterns *) * (maxPattern + 1));
        for (int i = 0; i <= maxPattern; i++) patternXref[i] = NULL;
        for (wpattern = patternHead; wpattern != NULL; wpattern = wpattern->next) {
            patternXref[wpattern->id] = wpattern;
        }
        ctr = last_seen = 0;
        for (htiter = itemUtilities->htItems->begin();
            htiter != itemUtilities->htItems->end(); htiter++) {
            wpatternList = new listT<genericPatterns>;

```

```

        utilities->showTime(1, &ctr, &last_seen, 10000);
        itemDef = htiter->second;
        sqlite3_reset(stmt_selItemPatterns);
        sqlite3_bind_int(stmt_selItemPatterns, 1, itemDef->itemid);
        while ((rc = sqlite3_step(stmt_selItemPatterns)) == SQLITE_ROW) {
            uint32_t wpatternid = sqlite3_column_int(stmt_selItemPatterns, 0);
            wpattern = patternXref[wpatternid];
            if (wpattern != NULL) wpatternList->insertRear(wpattern);
        }
        wpatternList->insertRear(starPattern);
        itemPatterns[itemDef->itemid] = wpatternList;
    }
}
else {
    printf("\tcreating new item/patterns\n");
    ctr = last_seen = 0;
    FILE *itempatterns;
    fopen_s(&itempatterns, ".\\itemPatterns.out", "a");
    for (htiter = itemUtilities->htItems->begin();
        htiter != itemUtilities->htItems->end(); htiter++) {
        utilities->showTime(1, &ctr, &last_seen, 100000);
        itemDef = htiter->second;
        wpatternList = new listT<genericPatterns>;

        for (wpattern = patternHead; wpattern != NULL; wpattern = wpattern->next)
            if (utilities->match(wpattern->pattern, itemDef->itemName)) {
                wpatternList->insertRear(wpattern);

                fprintf(itempatterns, "%u|%u\n", itemDef->itemid, wpattern->id);
            }
        }
        wpatternList->insertRear(starPattern);
        itemPatterns[itemDef->itemid] = wpatternList;
        if (parameters->debugParms >= 5) {
            fprintf(parameters->logFile, "item %i\t\t", itemDef->itemid);
            for (void * liter = wpatternList->getFront();
                liter != NULL; liter = wpatternList->revIterate(liter)) {
                fprintf(parameters->logFile, "%s\t\t",
                    wpatternList->lvalue(liter)->pattern);
            }
            fprintf(parameters->logFile, "\n");
        }
    }
}

```

```

        fclose(itempatterns);
    }
    printf("Completed linking items to patterns\n");
}
}

uint32_t Patterns::getGenericMatch(uint32_t witemid, listT<uint32_t> **usedItems,
                                   uint8_t allowAdd) {
    // Purpose: match an item to a pattern and then pick an unused generic item
    //           associated with that pattern.  If none exists (and you're allowed),
    //           add another generic item

    htItemPatterns_i htiter;
    genericPatterns *patternPtr;
    uint32_t *newItem;
    void *liter;
    listT<genericPatterns> *wpatternList;

    //////////////////////////////////////

    // First check the patterns
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile, "**wgetgetGenericMatch    item = %i\n", witemid);

    // If not using item patterns, call getGenericMatch the old way
    if (!(parameters->useItemPatternLists)) {
        char *wname = itemUtilities->getItemName(generics, witemid);
        uint32_t retItem = getGenericMatch(wname, usedItems, allowAdd);
        free(wname);
        return retItem;
    }

    // Get the pattern associated with the item
    wpatternList = itemPatterns[witemid];

    // If not found
    if (wpatternList == NULL) {
        // Get the name of the item
        char *wname = itemUtilities->getItemName(generics, witemid);
        printf("Missing item/pattern hash table for item %i -- %s\n", witemid, wname);
        // Call the old getGenericMatch
        uint32_t retItem = getGenericMatch(wname, usedItems, allowAdd);
        free(wname);
    }
}

```



```

    // Return the item
    return retItem;
}

// If there is a matching pattern, iterate through the generic items associated with
for (liter = wpatternList->getFront(); liter != NULL;
     liter = wpatternList->revIterate(liter)) {
    patternPtr = wpatternList->lvalue(liter);
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile,
            "**wgetGenericMatch    matched = %s\n", patternPtr->pattern);

    // Look for an unused generic for that pattern.
    // If you find one, add it to the used list and return it
    if (!(patternPtr->itemList->isEmpty())) {
        if (usedItems != NULL) {
            if ((*usedItems)->isEmpty()) {
                (*usedItems)->insertRear(
                    (patternPtr->itemList)->lvalue((patternPtr->itemList)->getFront()))
                return *((patternPtr->itemList)->lvalue((patternPtr->itemList)->getFront()));
            }
            for (liter = patternPtr->itemList->getFront(); liter != NULL;
                 liter = patternPtr->itemList->revIterate(liter)) {
                newItem = patternPtr->itemList->lvalue(liter);
                if (!((*usedItems)->exists(newItem))) {
                    (*usedItems)->insertRear(newItem);
                    return *newItem;
                }
            }
        }
        else {
            liter = (patternPtr->itemList)->getFront();
            newItem = (patternPtr->itemList)->lvalue(liter);
            if (newItem != NULL) return *newItem; else return 0;
        }
    }

    // If you can't find one, add a new one (and add it to the used list)
    // and return it
    if (allowAdd) {
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile,
                "**getGenericMatch    adding generic \n");
        newItem = (uint32_t *)malloc(sizeof(uint32_t) * 1);
    }
}

```

```

        *newItem = generics->addGenericItem();
        (patternPtr->itemList)->insertRear(newItem);
        if (usedItems != NULL)
            (*usedItems)->insertRear((patternPtr->itemList)->lvalue(
                (patternPtr->itemList)->getFront()));
        return *newItem;
    }
}

return 0;

}

uint32_t Patterns::getGenericMatch(char *wpattern, listT<uint32_t> *usedItems) {
    // Purpose: Pick an unused item associated with that pattern

    genericPatterns *patternPtr;
    uint32_t *newItem;
    void *liter;

    //////////////////////////////////////

    patternPtr = patternHead;
    while (patternPtr != NULL) {
        if (!strcmp(wpattern, patternPtr->pattern)) {
            if (patternPtr->itemList->lempty()) return 0;
            if (usedItems->lempty()) {
                usedItems->insertRear((patternPtr->itemList)->lvalue(
                    (patternPtr->itemList)->getFront()));
                return *((patternPtr->itemList)->lvalue((patternPtr->itemList)->getFront(
            }
            for (liter = patternPtr->itemList->getFront(); liter != NULL;
                liter = patternPtr->itemList->revIterate(liter)) {
                newItem = patternPtr->itemList->lvalue(liter);
                if (!usedItems->exists(newItem)) {
                    usedItems->insertRear(newItem);
                    return *newItem;
                }
            }
            return 0;
        }
        if (patternPtr == starPattern) break;
        patternPtr = patternPtr->next;
    }
}

```

```

        if (patternPtr == NULL) patternPtr = starPattern;
    }
    return 0;
}

uint32_t Patterns::getGenericMatch(char *wname, listT<uint32_t> **usedItems,
                                   uint8_t allowAdd) {
    // Purpose: match an item to a pattern and then pick an unused generic
    //           item associated with that pattern.  If none exists (and
    //           you're allowed), add another generic item

    genericPatterns *patternPtr;
    uint32_t *newItem;
    void *liter;

    //////////////////////////////////////

    // First check the patterns
    if (wname != NULL) { // handles (among other things) trying to match loops
        // in theory, we should be able to do patterns with
        // loops, but only if we save the old pattern (because
        // figuring out a common new pattern will be difficult
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, "**wgetgetGenericMatch   name = %s\n", wname);
        patternPtr = patternHead;
        while (patternPtr != NULL) {
            if (utilities->match(patternPtr->pattern, wname)) {
                if (parameters->debugParms >= 6)
                    fprintf(parameters->logFile,
                        "**wgetgetGenericMatch   matched = %s\n", patternPtr->pattern);
                // Look for an unused generic for that pattern.
                // If you find one, add it to the used list and return it
                if ( !(patternPtr->itemList->lempy()) ) {
                    if (usedItems != NULL) {
                        if ((*usedItems)->lempy()) {
                            (*usedItems)->insertRear((patternPtr->itemList)->lvalue(
                                (patternPtr->itemList)->getFront()));
                            return *((patternPtr->itemList)->lvalue(
                                (patternPtr->itemList)->getFront()));
                        }
                    }
                    for (liter = patternPtr->itemList->getFront(); liter != NULL;
                        liter = patternPtr->itemList->revIterate(liter)) {
                        newItem = patternPtr->itemList->lvalue(liter);
                    }
                }
            }
            patternPtr = patternPtr->next;
        }
    }
}

```

```

        if (!((*usedItems)->exists(newItem))) {
            (*usedItems)->insertRear(newItem);
            return *newItem;
        }
    }
}
else {
    return *((patternPtr->itemList)->lvalue(
        (patternPtr->itemList)->getFront()));
}
}
// If you can't find one, add a new one (and add it to the used list)
// and return it
if (allowAdd) {
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile, "**getGenericMatch    adding generic \n");
    newItem = (uint32_t *)malloc(sizeof(uint32_t) * 1);
    *newItem = generics->addGenericItem();
    (patternPtr->itemList)->insertRear(newItem);
    if (usedItems != NULL)
        (*usedItems)->insertRear((patternPtr->itemList)->lvalue(
            (patternPtr->itemList)->getFront()));

    return *newItem;
}
}
patternPtr = patternPtr->next;
}
}

// Then add it to the global wildcard
if (parameters->debugParms >= 6)
    fprintf(parameters->logFile, "**wgetGenericMatch    using Star Pattern\n");
if (!(starPattern->itemList->lempty()) ) {
    // Look for an unused generic for the star pattern.
    // If you find one, add it to the used list and return it
    if (usedItems != NULL) {
        if (!((*usedItems)->lempty())) {
            (*usedItems)->insertRear((starPattern->itemList)->lvalue(
                (starPattern->itemList)->getFront()));
            return *((starPattern->itemList)->lvalue(
                (starPattern->itemList)->getFront()));
        }
    }
}

```

```

        for (liter = starPattern->itemList->getFront(); liter != NULL;
             liter = starPattern->itemList->revIterate(liter)) {
            newItem = starPattern->itemList->lvalue(liter);
            if (!((*usedItems)->exists(newItem))) {
                (*usedItems)->insertRear(newItem);
                return *newItem;
            }
        }
    }
    else {
        return *((starPattern->itemList)->lvalue((starPattern->itemList)->getFront()))
    }
}

// If you can't find one, add a new one (and add it to the used list) and
// return it
if (allowAdd) {
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile, "**getGenericMatch    adding generic \n");
    newItem = (uint32_t *)malloc(sizeof(uint32_t) * 1);
    *newItem = generics->addGenericItem();
    (starPattern->itemList)->insertRear(newItem);
    if (usedItems != NULL)
        (*usedItems)->insertRear((starPattern->itemList)->lvalue(
                                (starPattern->itemList)->getFront()));

    return *newItem;
}

return 0;
}

void Patterns::insertPattern(genericPatterns *newPattern) {
    // Purpose: add the pattern to the hash table for later writing

    genericPatterns *wptr, *wprev;
    uint32_t wlength;

    //////////////////////////////////////

    wlength = (uint32_t)strlen(newPattern->pattern);
    wptr = patternHead;

```

```

wprev = NULL;
while (wptr != NULL) {
    if (wlength > strlen(wptr->pattern)) {
        newPattern->next = wptr;
        if (wprev != NULL) wprev->next = newPattern;
        if (wprev == NULL) patternHead = newPattern;
        return;
    }
    wprev = wptr;
    wptr = wptr->next;
}

// end of list
newPattern->next = NULL;
if (wprev != NULL) wprev->next = newPattern;
if (patternHead == NULL) patternHead = newPattern;
}

void Patterns::writePatterns() {
    // Purpose: write the pattern from the hash table to the database

    genericPatterns *wpattern;
    void *liter;
    int rc;
    uint32_t wid;

    //////////////////////////////////////

    wpattern = patternHead;
    while (wpattern != NULL) {
        if (patternHead->inDB == 0) {
            sqlite3_reset(stmt_insPatterns);
            sqlite3_bind_text(stmt_insPatterns, 1, wpattern->pattern, -1, SQLITE_TRANSIENT);
            sqlite3_step(stmt_insPatterns);
            wid = sqlite3_last_insert_rowid(parameters->db);
            wpattern->inDB = 1;
        }
        else
            wid = wpattern->id;

        for (liter = wpattern->itemList->getFront(); liter != NULL;
             liter = wpattern->itemList->revIterate(liter)) {
            sqlite3_reset(stmt_selPatternGenerics2);

```

```

        sqlite3_bind_int(stmt_selPatternGenerics2, 1, wid);
        sqlite3_bind_int(stmt_selPatternGenerics2, 2,
                        *(wpattern->itemList->lvalue(liter)));
        rc = sqlite3_step(stmt_selPatternGenerics2);
        if (rc != SQLITE_ROW) {
            sqlite3_reset(stmt_insPatternGenerics);
            sqlite3_bind_int(stmt_insPatternGenerics, 1, wid);
            sqlite3_bind_int(stmt_insPatternGenerics, 2,
                            *(wpattern->itemList->lvalue(liter)));
            sqlite3_step(stmt_insPatternGenerics);
        }
    }
    wpattern = wpattern->next;
}

for (liter = starPattern->itemList->getFront(); liter != NULL;
     liter = starPattern->itemList->revIterate(liter)) {
    sqlite3_reset(stmt_selPatternGenerics2);
    sqlite3_bind_int(stmt_selPatternGenerics2, 1, starPattern->id);
    sqlite3_bind_int(stmt_selPatternGenerics2, 2,
                    *(starPattern->itemList->lvalue(liter)));
    rc = sqlite3_step(stmt_selPatternGenerics2);
    if (rc != SQLITE_ROW) {
        sqlite3_reset(stmt_insPatternGenerics);
        sqlite3_bind_int(stmt_insPatternGenerics, 1, starPattern->id);
        sqlite3_bind_int(stmt_insPatternGenerics, 2,
                        *(starPattern->itemList->lvalue(liter)));
        sqlite3_step(stmt_insPatternGenerics);
    }
}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void Patterns::getSeparator(char *wstring, char **separatorStart,
                           char **separatorEnd) {
    // Purpose: find the first instance of the separator in the string
    //           after you've got the first one, keep going till you find a
    //           non-separator, e.g.  sdfsd///dsf//sd/sfd would return 5 and 7

    char wcheck;
    uint8_t includePeriod = 1;

```

```

/////////////////////////////////////////////////////////////////

*separatorStart = wstring;
wcheck = **separatorStart;
while (wcheck != 0 && (wcheck != 47 /* / */ && wcheck != 92 /* \ */ &&
    (wcheck != 46 || !includePeriod) /* . */)) {
    (*separatorStart)++;
    wcheck = **separatorStart;
}

*separatorEnd = *separatorStart;
wcheck = **separatorEnd;
while (wcheck != 0 &&
    (wcheck == 47 /* / */ || wcheck == 92 /* \ */ ||
    (wcheck == 46 && includePeriod) /* . */)) {
    (*separatorEnd)++;
    wcheck = **separatorEnd;
}
if ((*separatorStart) != 0) (*separatorEnd)--; // it moves one past last spot

return;
}

void Patterns::getWords(std::list<std::string>*qNames, std::string itemName) {
    // Purpose:  separate the itemName into words and stick them in the queue

    std::string wname, wword;
    int s1s, s1e, s2s, s2e;
    uint32_t wlength;

    //////////////////////////////////////////////////////////////////////

    wname = itemName;
    s1s = 0;
    s1e = 0;
    s2s = (int) wname.length();
    s2e = (int)wname.length();

    while (s1s < wname.length()) {
        // find the location of the next separator
        if (s1e == 0)
            s2s = (int) wname.find_first_of("/\\.", s1e);
    }
}

```



```

        else
            s2s = (int) wname.find_first_of("/\\.", s1e + 1);
            if (s2s == std::string::npos) { s2e = s2s; }
            s2e = (int) wname.find_first_not_of("/\\.", s2s) - 1;

            // get the word between the current and next separator
            if (s2s != std::string::npos && s2s > s1e + 1) {
                wlength = s2s - s1e;
                wword = wname.substr(s1e + 1, wlength - 1);
                qNames->push_back(wword);

                s1s = s2s;
                s1e = s2e;
            }
            else {
                if (s1e >= 0) {
                    wword = wname.substr(s1e + 1);
                    qNames->push_back(wword);
                }
                break;
            }
        }
    }
}

uint32_t Patterns::numberCount(std::string wname) {
    // Purpose: count the number of numbers in the string
    uint32_t i;
    uint32_t nbrCount;

    //////////////////////////////////////

    if (wname.length() == 0) return 0;

    nbrCount = 0;
    for (i = 0; i < wname.length(); i++) {
        if (wname.at(i) >= '0' && wname.at(i) <= '9') nbrCount++;
    }
    return nbrCount;
}

uint32_t Patterns::insertWord(std::unordered_map<std::string, wordBox *> *htWordList,

```

```

std::string wname, uint32_t *wordCount, uint32_t group) {
// Purpose: insert the word into the unordered map; if it already exists,
// increase the count

htWordList_i htiter;
wordBox *wWord;
std::string tmpName;
tmpName = std::string(wname);

////////////////////////////////////

htiter = htWordList->find(tmpName);
if (htiter != htWordList->end()) {
    wWord = htiter->second;
    wWord->count++;
    return wWord->id;
}
wWord = (wordBox *)malloc(sizeof(wordBox) * 1);
if (wWord == NULL) {
    printf("**insertWord -- unable to allocate newWord; returning\n");
    return 0;
}
wWord->id = (*wordCount)++;
wWord->group = group;
wWord->name = new std::string("hello"); //wname;
wWord->name->assign(wname);
wWord->length = (uint32_t)wname.length();
wWord->count = 1;

(*htWordList)[tmpName] = wWord;

return wWord->id;
}

std::string Patterns::getPhrase(wordBox language[], uint32_t wordCount,
                                uint32_t *ids) {
// Purpose: Create a phrase (pattern) from the words (ids) passed in

char wphraseConst[2000];
std::string retPhrase;
uint32_t prtStar, i;

////////////////////////////////////

```

```

sprintf_s(wphraseConst, 2000, "^");
prtStar = 0;
for (i = 1; i < wordCount; i++) {
    if (ids[i] > 1) {
        if (i > 1)
            sprintf_s(wphraseConst, 2000, "%s/%s", wphraseConst,
                    language[ids[i] - 1].name->c_str());
        else
            sprintf_s(wphraseConst, 2000, "%s%s", wphraseConst,
                    language[ids[i] - 1].name->c_str());
        prtStar = 0;
    }
    else if (!prtStar) {
        prtStar = 1;
        if (i > 1)
            sprintf_s(wphraseConst, 2000, "%s/.*", wphraseConst);
        else
            sprintf_s(wphraseConst, 2000, "%s.*", wphraseConst);
    }
}
sprintf_s(wphraseConst, 2000, "%s$", wphraseConst);

retPhrase = std::string(wphraseConst);

return retPhrase;
}

uint32_t Patterns::insertPhrase(wordBox **head, std::string wname,
                                uint32_t *windex) {
    // Purpose: insert the phrase into the linked list

    // Using wordcount for two things:
    //   not null -- return the number of words in the linked list
    //   null      -- allow duplicate keys
    // if windex is not null, it's the key; otherwise, name is the key
    wordBox *wptr, *wprev, *newWord;

    //////////////////////////////////////

    wptr = *head;
    wprev = NULL;
    while (wptr != NULL) {

```

```

if ((windex != NULL && *windex < wptr->id) ||
    ((wptr->name)->compare(wname) > 0)) { //wptr->name > wname
newWord = (wordBox *)malloc(sizeof(wordBox) * 1);
if (newWord == NULL) {
    printf("**insertPhrase -- unable to allocate newWord; returning\n");
    return 0;
}
newWord->id = (windex != NULL ? *windex : 0);
newWord->group = 0;
newWord->name = new std::string("hello"); // wname;
newWord->name->assign(wname);
newWord->length = (uint32_t)wname.length();
newWord->count = (windex != NULL ? *windex : 0);
newWord->prev = wptr->prev;
newWord->next = wptr;
wptr->prev = newWord;
if (newWord->prev != NULL)
    (newWord->prev)->next = newWord;
else
    *head = newWord;
if (parameters->debugParms >= 4)
    fprintf(parameters->logFile, "\t\tinserting  %s\n",
            newWord->name->c_str());
if (windex != NULL && newWord->id > *windex)
    printf("a*****id = %i  \n", newWord->id);
return newWord->id;
}
wprev = wptr;
wptr = wptr->next;
}

// end of list
newWord = (wordBox *)malloc(sizeof(wordBox) * 1);
if (newWord == NULL) {
    printf("**insertPhrase -- unable to allocate newWord; returning\n");
    return 0;
}
newWord->id = (windex != NULL ? *windex : 0);
newWord->group = 0;
newWord->name = new std::string("hello"); // wname;
newWord->name->assign(wname);
newWord->length = (uint32_t)wname.length();
newWord->count = (windex != NULL ? *windex : 0);

```

```

newWord->prev = wprev;
newWord->next = NULL;
if (newWord->prev != NULL)
    (newWord->prev)->next = newWord;
else
    *head = newWord;

if (parameters->debugParms >= 4)
    fprintf(parameters->logFile, "\t\tinserting %s\n", newWord->name->c_str());
return newWord->id;
}

void Patterns::createPatterns() {
    // Purpose: create patterns

    //Variables to break the item table into a manageable size
    uint32_t loopCount;
    double wperc;

    // Variables to track and report elapsed time
    uint32_t ctr, last_seen;
    uint32_t wordCount;
    uint32_t phraseCount;

    // Variables to insert patterns into the database
    char exec_string[2000];

    uint32_t i, j, ctr2;
    int rc;
    uint32_t wordId; // used to avoid too many brackets
    uint32_t hasZero;
    uint32_t starCount; // number of asterisks in the name

    // Variables to determine which patterns to save
    uint32_t threshold;
    uint32_t checkNbr;
    uint32_t keepers, totKeepers;

    // the dataset -- every item name in the item table
    itemBox *dataset;
    uint32_t datasetLength;

    // all of the "words" in the dataset (i.e., everything between consecutive '/'s

```

```

wordBox *language;
wordBox *wordPtr, *finalPhrases; // sorted list of words -- precursor to language

                                // summary statistics
uint32_t* maxFreqLength;
uint32_t* maxWords;           // maximum number of words in a record
uint32_t maxLength;          // maximum number of non-zero ids in a record
                                // (some words may have their id zeroed out

// a pattern phrase made up of words
std::string wphrase;

// Variables to create a hash table of potential patterns
uint32_t *saveIds;
uint32_t *hashid2;
uint64_t whash64;
uint32_t isCollision;
int last1;
std::unordered_map<uint64_t, phraseType *> *htPhrases;
std::unordered_map<uint64_t, phraseType *>::const_iterator htiter;

std::unordered_map<std::string, wordBox *> *wordHead;
std::unordered_map<std::string, wordBox *>::const_iterator htWordIter;
wordBox *wWord;

phraseType *xphrase;

std::list<std::string> *qNames;
std::string wName;

uint8_t foundIt;

uint32_t wordLength;
uint32_t highMark;
uint32_t itemsRemoved;

uint32_t maxGroup;
bool isFiles;

sqlite3_stmt *stmt_selItem, *stmt_selItem1, *stmt_selItem2;
sqlite3_stmt *stmt_selPatterns, *stmt_insPatterns;

```

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

```
// Initialize variables
```

```
maxGroup = 57;
```

```
qNames = new std::list<std::string>;
```

```
dataset = (itemBox *)malloc(sizeof(itemBox) * parameters->maxItemid);
```

```
htPhrases = new std::unordered_map<uint64_t, phraseType *>;
```

```
wordHead = new std::unordered_map<std::string, wordBox_t *>;
```

```
finalPhrases = NULL;
```

```
wordCount = 0;
```

```
// Create the SQL statements
```

```
sprintf_s(exec_string, 2000, "SELECT id FROM patterns WHERE pattern like ?1;");
```

```
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selPatterns, 0);
```

```
if (rc != SQLITE_OK) printf("**Error** couldn't prepare Select Pattern statement\n");
```

```
sprintf_s(exec_string, 2000, "insert INTO patterns (pattern) VALUES(?1);");
```

```
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_insPatterns, 0);
```

```
if (rc != SQLITE_OK) {
```

```
    printf("Error: %i. couldn't prepare Insert Patterns Statement\n", rc);
```

```
    return;
```

```
}
```

```
sprintf_s(exec_string, 2000, "SELECT items.itemid, items.itemtypeid, files.filename
```

```
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selItem1, 0);
```

```
if (rc != SQLITE_OK) printf("**Error** couldn't prepare Select Item statement\n");
```

```
sprintf_s(exec_string, 2000,
```

```
    "SELECT items.itemid, items.itemtypeid, keys.keyname FROM items, keys where ite
```

```
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selItem2, 0);
```

```
if (rc != SQLITE_OK) printf("**Error** couldn't prepare Select Item statement\n");
```

```
printf("Loading %i Items\n",parameters->maxItemid);
```

```
printf("\tloading files\n");
```

```
i = 0; last_seen = 0;
```

```
stmt_selItem = stmt_selItem1;
```

```
rc = sqlite3_step(stmt_selItem);
```

```
isFiles = 1;
```

```
while (rc == SQLITE_ROW) {
```

```
    // Report progress
```

```
    if (utilities->showTime(0, &i, &last_seen, parameters->increment))
```

```
        printf("\t\tWordcount = %i\n", (int)wordCount);
```

```

dataset[i - 1].itemid = sqlite3_column_int(stmt_selItem, 0);
dataset[i - 1].itemtypeid = sqlite3_column_int(stmt_selItem, 1);
dataset[i - 1].name = new std::string("hello");
dataset[i - 1].name->assign((const char *) sqlite3_column_text(stmt_selItem, 2),
dataset[i - 1].wordCount = 0;
dataset[i - 1].group = getGroup((char *)sqlite3_column_text(stmt_selItem, 2),
                                dataset[i - 1].itemtypeid);
if (dataset[i - 1].group > maxGroup)
    printf("PROBLEM!! --- GROUP %i ASSIGNED to record %i\n",
          dataset[i - 1].group,i - 1);

// Get the words from the record
qNames->clear();
getWords(qNames, *(dataset[i - 1].name));
dataset[i - 1].wordCount = (uint32_t) qNames->size();
dataset[i - 1].words = (uint32_t *)
    malloc(sizeof(uint32_t) * dataset[i - 1].wordCount);
for (ctr2 = 0; ctr2 < dataset[i - 1].wordCount; ctr2++)
    dataset[i - 1].words[ctr2] = 0;

// Add the words in the record into a list of words but if the words have
// an exclamation point or more than 2 numbers, leave the wordid as zero
ctr2 = 0;
while (!(qNames->empty())) {
    wName = qNames->front();
    qNames->pop_front();
    checkNbr = numberCount(wName);
    if ((checkNbr < 3) && (wName.find_first_of('!') == std::string::npos)) {
        wordId = insertWord(wordHead, wName, &wordCount, dataset[i - 1].group);
        dataset[i - 1].words[ctr2++] = wordId;
    }
    else
        dataset[i - 1].words[ctr2++] = 0;
}

if (i == parameters->maxItemid) {
    printf("too many items\n"); break;
}
rc = sqlite3_step(stmt_selItem);
if (rc != SQLITE_ROW && isFiles) {
    printf("\tloading keys\n");
    isFiles = 0;
    sqlite3_finalize(stmt_selItem);
}

```



```

        stmt_selItem = stmt_selItem2;
        rc = sqlite3_step(stmt_selItem);
    }
}
sqlite3_finalize(stmt_selItem);
datasetLength = i;

// Once the dataset is loaded, convert the sorted linked list of words into
// an array for faster processing
printf("Converting item list into Language array\n");
language = (wordBox *)malloc(sizeof(wordBox) * wordCount);
maxFreqLength = (uint32_t *)malloc(sizeof(uint32_t) * (maxGroup + 1));
for (i = 0; i <= maxGroup; i++) maxFreqLength[i] = 0;

for (htWordIter = wordHead->begin(); htWordIter != wordHead->end(); htWordIter++)
    wWord = htWordIter->second;
    language[wWord->id].id = wWord->id;
    language[wWord->id].count = wWord->count;

    // wWord->name, note: I'm not freeing it when I free wordHead
    language[wWord->id].name = new std::string("hello");
    language[wWord->id].name = wWord->name;
    language[wWord->id].length = wWord->length;
    if ((wWord->length * wWord->count) > maxFreqLength[wWord->group])
        maxFreqLength[wWord->group] = wWord->length * wWord->count;
}
delete wordHead;

// remove from the dataset any words that only occur once
printf("Removing singletons from dataset\n");
maxLength = 0;
maxWords = (uint32_t *)malloc(sizeof(uint32_t) * (maxGroup + 1));
for (i = 0; i <= maxGroup; i++) maxWords[i] = 0;
ctr2 = last_seen = itemsRemoved = 0;
for (i = 0; i < datasetLength; i++) {
    if (utilities->showTime(0, &ctr2, &last_seen, parameters->increment))
        printf("\t\t%i singletons removed\n", (int)itemsRemoved);
    ctr = 0;
    for (j = 0; j < dataset[i].wordCount; j++) {
        wordId = dataset[i].words[j];
        if (wordId != 0 && language[wordId].count < 2) {
            itemsRemoved++;
            dataset[i].words[j] = 0;
        }
    }
}

```

```

    }
    if (dataset[i].words[j] != 0) ctr++;
}
if (ctr > maxLength) maxLength = ctr;
if (dataset[i].wordCount > maxWords[dataset[i].group])
    maxWords[dataset[i].group] = dataset[i].wordCount;
}

printf("Building possible patterns\n");

totKeepers = 0;
for (loopCount = 0; loopCount < maxGroup; loopCount++) {
    wperc = getPerc(loopCount);
    threshold = (uint32_t)(wperc * (double)maxFreqLength[loopCount]);

    // Build a list of all possible patterns from the dataset
    printf("Building all possible patterns for Loop %i\n", loopCount);
    ctr = 0;
    last_seen = 0;
    phraseCount = 0;
    saveIds = (uint32_t *)malloc(sizeof(uint32_t) * (maxWords[loopCount] + 1));
    hashid2 = (uint32_t *)malloc(sizeof(uint32_t) * (maxWords[loopCount] + 1));
    htPhrases->clear();
    for (i = 0; i < datasetLength; i++) {
        // Report on progress
        if (utilities->showTime(0, &ctr, &last_seen, parameters->increment))
            printf("\t\tPhrasecount = %i\n", (int)phraseCount);

        // Only look at records with words
        if (dataset[i].wordCount > 0 && dataset[i].group == loopCount) {
            highMark = min(dataset[i].wordCount, maxWords[loopCount]);
            if (highMark > 15) highMark = 15;
            // First, make a copy of the record's list of words
            for (j = 0; j < highMark; j++) saveIds[j] = (dataset[i].words[j]);

            // Now get every possible pattern from the record by
            // "turning on/off" the words like incrementing a binary number.
            // Put them in a hash table and keep count of how often they occur
            last1 = highMark - 1; // start at the right end
            while (last1 > -1) { // while you're not all the way to the left
                if (saveIds[last1] != 0) { // if the current word is on

```

```

// turn the current word off and
// every word to the right of it on
saveIds[last1] = 0;
for (j = last1 + 1; j < highMark; j++) {
    saveIds[j] = (dataset[i].words[j]);
}

// find it in the phrase hash table and update the count
// note that since g_hash_table_insert free's the value
// each time, you're always doing a new insert
hasZero = 0;
wordLength = 1;
for (j = 0; j < highMark; j++) {
    if (saveIds[j] != 0 || !hasZero) wordLength++;
    hasZero = (saveIds[j] == 0 ? 1 : 0);
}

hasZero = 0;
hashid2[0] = 1;
wordLength = 1;
for (j = 0; j < highMark; j++) {
    if (saveIds[j] != 0 || !hasZero)
        hashid2[wordLength++] = saveIds[j] + 1;
    hasZero = (saveIds[j] == 0 ? 1 : 0);
}

foundIt = 0;
whash64 = utilities->hashword64((const uint32_t *)hashid2,
                                wordLength, 0xfacedeafdeafce);
while (1) {
    htiter = htPhrases->find(whash64);
    if (htiter == htPhrases->end()) break;
    xphrase = htiter->second;
    if (xphrase->wordCount == wordLength) {
        isCollision = 0;
        for (j = 0; j < wordLength; j++) {
            if (xphrase->ids[j] != hashid2[j]) {
                isCollision = 1;
                break;
            }
        }
        if (!isCollision) { foundIt = 1; break; }
    }
    whash64++;
}

```

```

    }

    if (foundIt == 0) {
        xphrase = (phraseType *)malloc(sizeof(phraseType) * 1);
        xphrase->ids = (uint32_t *)
            malloc(sizeof(uint32_t) * wordLength);
        xphrase->wordCount = wordLength;
        xphrase->length = 0;
        for (j = 0; j < wordLength; j++) {
            xphrase->ids[j] = hashid2[j];
            if (hashid2[j] > 1 && j > 0)
                xphrase->length += language[hashid2[j] - 1].length;
        }
        xphrase->count = 1;
        phraseCount++;
        (*htPhrases)[whash64] = xphrase;
    }
    else {
        (xphrase->count)++;
    }
}

// move the pointer all the way to the right
last1 = highMark - 1;
// and move to the right looking for the first word that
// is "turned on"
while ((last1 >= 0) && (saveIds[last1] == 0))
    last1 = last1 - 1;
}

}

free(saveIds);
free(hashid2);

// Iterate through the hash table of patterns
printf("Searching for relevant patterns -- phraseCount is %i\n",
       (int)phraseCount );

keepers = 0;
ctr = 0;
last_seen = 0;
for (htiter = htPhrases->begin(); htiter != htPhrases->end(); htiter++) {
    xphrase = htiter->second;
    if (utilities->showTime(0, &ctr, &last_seen, (parameters->increment) * 100))

```

```

        printf("\t\tKeepers = %i\n", (int)keepers);

    if (xphrase->length * xphrase->count > threshold) {
        wphrase = getPhrase(language, xphrase->wordCount, xphrase->ids);

        keepers++;
        totKeepers++;
        printf("Score = %u (%f)      count = %8u      length = %3u      phrase = %s\n",
               xphrase->length * xphrase->count,
               ((double)(xphrase->length * xphrase->count)) /
               ((double)maxFreqLength[loopCount]),
               xphrase->count, xphrase->length, wphrase.c_str());

        // Sort by the number of actual words in the phrase descending
        starCount = 0;
        for (j = 0; j < wphrase.length(); j++) {
            if (wphrase.at(j) == '*') starCount++;
        }
        if (starCount > xphrase->wordCount) {
            fprintf(parameters->logFile, "starCount = %u\n", starCount);
            starCount = xphrase->wordCount;
        }
        wordCount = xphrase->wordCount - starCount;
        insertPhrase(&finalPhrases, wphrase, &wordCount);
    }
    free(xphrase->ids);
    free(xphrase);
}
printf("Keeping %u patterns\n", keepers);

printf("Removing phrases\n");
htPhrases->clear();
}
printf("Keeping %u total patterns\n", totKeepers);

while (finalPhrases != NULL) {
    sqlite3_reset(stmt_selPatterns);
    sqlite3_bind_text(stmt_selPatterns, 1, finalPhrases->name->c_str(), -1,
                      SQLITE_TRANSIENT);
    rc = sqlite3_step(stmt_selPatterns);

    if (rc != SQLITE_ROW) {
        sqlite3_reset(stmt_insPatterns);
    }
}

```

```

        sqlite3_bind_text(stmt_insPatterns, 1, finalPhrases->name->c_str(), -1,
                           SQLITE_TRANSIENT);
        sqlite3_step(stmt_insPatterns);
    }
    wordPtr = finalPhrases;
    finalPhrases = finalPhrases->next;
    free(wordPtr);
}

}

uint32_t Patterns::getGroup(char *name, uint32_t itemtype) {
    // Purpose: programatically divide the total items into groups smaller enough
    // to handle
    if (itemtype != I_FILE && itemtype != I_KEY) return 1;

    if (itemtype == I_KEY && utilities->matchu("^C.*", name)) return 2;
    if (itemtype == I_KEY && utilities->matchu("^S.*", name)) return 3;
    if (itemtype == I_KEY) return 48;

    if (utilities->matchu("^.: /WINDOWS/B.*", name)) return 4;
    if (utilities->matchu("^.: /WINDOWS/C.*", name)) return 5;
    if (utilities->matchu("^.: /WINDOWS/D.*", name)) return 6;
    if (utilities->matchu("^.: /WINDOWS/E.*", name)) return 7;
    if (utilities->matchu("^.: /WINDOWS/F.*", name)) return 8;
    if (utilities->matchu("^.: /WINDOWS/G.*", name)) return 9;
    if (utilities->matchu("^.: /WINDOWS/H.*", name)) return 10;
    if (utilities->matchu("^.: /WINDOWS/INFUSEDAPPS/PACKAGES/MICROSOFT.W.*", name)) return 11;
    if (utilities->matchu("^.: /WINDOWS/INFUSEDAPPS/PACKAGES/.*", name)) return 50;
    if (utilities->matchu("^.: /WINDOWS/I.*", name)) return 11;
    if (utilities->matchu("^.: /WINDOWS/J.*", name)) return 12;
    if (utilities->matchu("^.: /WINDOWS/K.*", name)) return 13;
    if (utilities->matchu("^.: /WINDOWS/L.*", name)) return 14;
    if (utilities->matchu("^.: /WINDOWS/N.*", name)) return 15;
    if (utilities->matchu("^.: /WINDOWS/O.*", name)) return 16;
    if (utilities->matchu("^.: /WINDOWS/P.*", name)) return 17;
    if (utilities->matchu("^.: /WINDOWS/Q.*", name)) return 18;
    if (utilities->matchu("^.: /WINDOWS/R.*", name)) return 19;
    if (utilities->matchu("^.: /WINDOWS/S.*", name)) return 20;
    if (utilities->matchu("^.: /WINDOWS/T.*", name)) return 21;
    if (utilities->matchu("^.: /WINDOWS/U.*", name)) return 22;
    if (utilities->matchu("^.: /WINDOWS/V.*", name)) return 23;
    if (utilities->matchu("^.: /WINDOWS/W.*", name)) return 24;

```

```

if (utilities->matchu("^.: /WINDOWS/X.*", name)) return 25;
if (utilities->matchu("^.: /WINDOWS/Y.*", name)) return 26;
if (utilities->matchu("^.: /WINDOWS/Z.*", name)) return 27;
if (utilities->matchu("^.: /WINDOWS/A.*", name)
    && !utilities->matchu("^.: /WINDOWS/ASSEMBLY/GAC_MSIL.*", name)) return 28;
if (utilities->matchu("^.: /WINDOWS/M.*", name)
    && !utilities->matchu("^.: /WINDOWS/MICROSOFT.NET/ASSEMBLY/GAC_MSIL.*", name)
    && !utilities->matchu("^.: /WINDOWS/MICROSOFT.NET/Framework64.*", name)) return
if (utilities->matchu("^.: /WINDOWS/MICROSOFT.NET/Framework64.*", name)) return 30;

if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/B.*", name)) return 31;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/G.*", name)) return 32;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/M.*", name)) return 33;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/O.*", name)) return 34;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/P.*", name)) return 35;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/S.*", name)) return 36;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/W.*", name)) return 37;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/X.*", name)) return 38;
if (utilities->matchu("^.: /PROGRAM FILES/WINDOWSAPPS/.*", name)) return 39;
if (utilities->matchu("^.: /PROGRAM FILES/.*", name)) return 40;

if (utilities->matchu("^.: /PROGRAM FILES (X86)/COMMON FILES/.*", name)) return 41;
if (utilities->matchu("^.: /PROGRAM FILES (X86)/MICROSOFT OFFICE/OFFICE15/.*", name)
if (utilities->matchu("^.: /PROGRAM FILES (X86)/MICROSOFT OFFICE/.*", name)) return
if (utilities->matchu("^.: /PROGRAM FILES (X86)/W.*", name)) return 44;
if (utilities->matchu("^.: /PROGRAM FILES (X86)/.*", name)) return 45;

if (utilities->matchu("^.: /$ORPHANFILES/.*", name)) return 46;

if (utilities->matchu("^.: /PROGRAMDATA/.*", name)) return 51;
if (utilities->matchu("^.: /USERS/./APPDATA/LOCAL/MOZILLA/.*", name)) return 52;
if (utilities->matchu("^.: /USERS/./APPDATA/LOCAL/MICROSOFT/.*", name)) return 53;
if (utilities->matchu("^.: /USERS/./APPDATA/LOCAL/CHROME/.*", name)) return 54;
if (utilities->matchu("^.: /USERS/./APPDATA/LOCAL/PACKAGES/.*", name)) return 55;
if (utilities->matchu("^.: /USERS/./APPDATA/LOCAL/TEMP/.*", name)) return 56;
if (utilities->matchu("^.: /USERS/./APPDATA/.*", name)) return 57;

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "**getGroups** cats & dogs = %s\n", name);

return 47;
}

```

```

double Patterns::getPerc(uint32_t wgroup) {
    // Purpose: programatically create a threshold that a percent of the total
    if (wgroup == 1) return 0.05; //not keys or files
    if (wgroup == 2) return 1.70; // 1.50; //keys/c
    if (wgroup == 3) return 5.00; // 4.00; //keys/s
    if (wgroup == 48) return 4.00; // 3.70; //keys
    if (wgroup == 29 || wgroup == 30) return 2.10; // 1.30; // windows/m
    if (wgroup == 28) return 1.60; // 0.75; // windows/a
    if (wgroup == 4) return 1.00; // 0.70; // windows/b
    if (wgroup == 5) return 0.95; // 0.85; // windows/c
    if (wgroup == 6) return 0.60; // windows/d
    if (wgroup == 7) return 0.50; // windows/e
    if (wgroup == 9) return 0.35; // windows/g
    if (wgroup == 10) return 2.00; // 1.50; // windows/h
    if (wgroup == 11) return 0.80; // windows/i
    if (wgroup == 12) return 0.10; // windows/j
    if (wgroup == 13) return 0.10; // windows/j
    if (wgroup == 49) return 1.90; // 1.20; // windows/infusedapps/packages/microsoft
    if (wgroup == 50) return 1.00; // 0.80; // windows/infusedapps/packages
    if (wgroup == 14) return 0.70; // 0.45; // windows/l
    if (wgroup == 15) return 0.10; // windows/n
    if (wgroup == 16) return 0.60; // 0.40; // windows/p
    if (wgroup == 17) return 0.50; // 0.40; // windows/p
    if (wgroup == 18) return 0.10; // windows/q
    if (wgroup == 19) return 0.50; // windows/r
    if (wgroup == 20) return 0.50; // 0.40; // windows/s
    if (wgroup == 21) return 0.40; // windows/t
    if (wgroup == 22) return 0.10; // windows/t
    if (wgroup == 23) return 1.00; // 0.50; // windows/v
    if (wgroup == 24) return 0.90; // 0.35; // windows/w
    if (wgroup == 25) return 0.10; // windows/x
    if (wgroup == 26) return 0.10; // windows/y
    if (wgroup == 27) return 0.10; // windows/z

    if (wgroup == 31) return 0.80; //program files/windowsapps/b
    if (wgroup == 32) return 0.60; //program files/windowsapps/g
    if (wgroup == 33) return 0.50; //program files/windowsapps/m
    if (wgroup == 34) return 0.80; //program files/windowsapps/o
    if (wgroup == 35) return 0.80; //program files/windowsapps/p
    if (wgroup == 36) return 0.80; //program files/windowsapps/s
    if (wgroup == 37) return 0.25; //program files/windowsapps/w
    if (wgroup == 38) return 0.30; //program files/windowsapps/x
    if (wgroup == 39) return 3.00; // 1.20; // 0.50; //program files/windowsapps/

```



```

if (wgroup == 40) return 0.60; // 0.50; //program files/

if (wgroup == 41) return 0.50; // x86/common files
if (wgroup == 42) return 0.50; // x86/office/office15
if (wgroup == 43) return 2.00; // 0.35; // x86/office/
if (wgroup == 44) return 0.55; // 0.50; // x86/w
if (wgroup == 45) return 1.00; // 0.60; // x86 everything else

if (wgroup == 46) return 0.50; // /$orphanFiles
if (wgroup == 47) return 2.00; // everything else
if (wgroup == 51) return 1.00; // 0.50; /programdata
if (wgroup == 52) return 7.00; // 6.00; /users././appdata/local/mozilla
if (wgroup == 53) return 1.5; // 0.8; // 0.30; // /users././appdata/local/microso
if (wgroup == 55) return 4.50; // 3.00; /users././appdata/local/packages
if (wgroup == 56) return 4.00; // 1.00; /users././appdata/local/temp

return 0.25;
}

```

6.5 Terms

Header.

```
#pragma once
#include <unordered_map>
#include "TearCPP.h"
#include "Patterns.h"
#include "Generics.h"
#include "TermList.h"
#include "Utilities.h"
#define htTerms_i std::unordered_map<uint32_t, int>::const_iterator
#define htTXref_i std::unordered_map<int, term_entry>::const_iterator

class Terms {

    typedef struct term_type {
        uint32_t actor;
        uint32_t action;
        uint32_t appexec;
        uint32_t itemtype;
        uint32_t item;
        uint8_t isString;
        int id;
    } term_entry;

    sqlite3_stmt *stmt_selRevTrm;
    sqlite3_stmt *stmt_insTrm;
    sqlite3_stmt *stmt_insStrTrm;
    sqlite3_stmt *stmt_selAllTrm;

    int maxTerm;

    std::unordered_map<int, term_entry> htTXref;
    std::unordered_map<uint32_t, int> htTerms;

    Utilities *utilities;
    Parameters *parameters;
    Patterns *patterns;
    Generics *generics;

    void Terms::addTerm(int wterm);
    int Terms::getTermT(uint32_t wactor, uint32_t waction, uint32_t wappexec,
        uint32_t witemtype, uint32_t witem,
```

```

        uint8_t useGeneric, uint8_t allowAdd);
int Terms::getNewTermT(int wterm, uint32_t wgeneric, uint8_t allowAdd);
int TermLookup(uint32_t wactor, uint32_t waction, uint32_t wappexec,
               uint32_t witemtype, uint32_t witem);
public:
    Terms(Utilities *wutilities, Parameters *wparameters, Patterns *wpatterns,
          Generics *wgenerics);
    ~Terms();

    uint8_t initialize() {
        int rc = sqlite3_reset(stmt_selAllTrm);
        if (rc == SQLITE_OK)
            return 1;
        else
            return 0;
    }
    int iterate() {
        int rc = sqlite3_step(stmt_selAllTrm);
        if (rc == SQLITE_ROW) return sqlite3_column_int(stmt_selAllTrm, 0);
        else return -1;
    }
    void loadTerms();
    int addStringTerm(int wterm);

    uint32_t getActor(int wterm);
    uint32_t getAction(int wterm);
    uint32_t getAppexec(int wterm);
    uint32_t getItemtype(int wterm);
    uint32_t getItem(int wterm);
    uint8_t isString(int wterm);

    int getMaxTerm() { return maxTerm; }

    uint32_t getActor(TermList *wterm) { return getActor(wterm->getTerm()); }
    uint32_t getAction(TermList * wterm) { return getAction(wterm->getTerm()); }
    uint32_t getAppexec(TermList * wterm) { return getAppexec(wterm->getTerm()); }
    uint32_t getItemtype(TermList * wterm) { return getItemtype(wterm->getTerm()); }
    uint32_t getItem(TermList * wterm) { return getItem(wterm->getTerm()); }
    uint8_t isString(TermList * wterm) { return isString(wterm->getTerm()); }

    int getNewTerm(int wterm, uint32_t wgeneric) {
        return getNewTermT(wterm, wgeneric, 0);
    }
}

```

```

int getNewTermA(int wterm, uint32_t wgeneric) {
    return getNewTermT(wterm, wgeneric, 1);
}

int getTerm(uint32_t wactor, uint32_t waction, uint32_t wappexec,
            uint32_t witemtype, uint32_t witem) {
    return getTermT(wactor, waction, wappexec, witemtype, witem, 0, 0);
}

int getTermA(uint32_t wactor, uint32_t waction, uint32_t wappexec,
            uint32_t witemtype, uint32_t witem) {
    return getTermT(wactor, waction, wappexec, witemtype, witem, 0, 1);
}

int getGenericTerm(uint32_t wactor, uint32_t waction, uint32_t wappexec,
            uint32_t witemtype, uint32_t witem) {
    return getTermT(wactor, waction, wappexec, witemtype, witem, 1, 0);
}

int getGenericTermA(uint32_t wactor, uint32_t waction, uint32_t wappexec,
            uint32_t witemtype, uint32_t witem) {
    return getTermT(wactor, waction, wappexec, witemtype, witem, 1, 1);
}

};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Terms                                                                                       //
// Notes:                                                                                           //
//     1. Terms are replications of activities where the items are replaced //
//         by generics, i.e., variables that allow generics to be assigned //
//         multiple items.                                                                           //
// Initial Release: 29-Sep-17                                                                       //
//                                                                                               //
// Revision History:                                                                                //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "Terms.h"

Terms::Terms(Utilities *wutilities, Parameters *wparameters,
            Patterns *wpatterns, Generics *wgenerics) {

```

```

char exec_string[2000];
int rc;
sqlite3_stmt *stmt;

utilities = wutilities;
parameters = wparameters;
generics = wgenerics;
patterns = wpatterns;
sprintf_s(exec_string, 2000, "SELECT id from terms where actor = ?1 and action = ?";
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_selRevTrm), 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Terms Statement\n", rc);
}

sprintf_s(exec_string, 2000, "INSERT INTO terms (actor, action, appexec, itemtype, item, itemstring) VALUES (?1, ?2, ?3, ?4, ?5, ?6)";
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_insTrm), 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Terms Statement\n", rc);
}

sprintf_s(exec_string, 2000, "INSERT INTO terms (actor, action, appexec, itemtype, item, itemstring) VALUES (?1, ?2, ?3, ?4, ?5, ?6)";
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_insStrTrm), 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert String Terms Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT id, actor, action, appexec, itemtype, item, itemstring from terms";
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_selAllTrm), 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Terms Statement\n", rc);
}

sprintf_s(exec_string, 2000, "SELECT MAX(id) FROM terms;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt), 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Get MaxTerm ID Statement\n", rc);
}
rc = sqlite3_step(stmt);
maxTerm = (rc == SQLITE_ROW ? sqlite3_column_int(stmt, 0) + 1 : 1);
}

Terms::~Terms() {}

```

```

uint8_t Terms::isString(int wterm) {
    htTXref_i htiter;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.isString;
}

uint32_t Terms::getActor(int wterm) {
    htTXref_i htiter;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.actor;
}

uint32_t Terms::getAction(int wterm) {
    htTXref_i htiter;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.action;
}

uint32_t Terms::getAppexec(int wterm) {
    htTXref_i htiter;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.appexec;
}

uint32_t Terms::getItemtype(int wterm) {
    htTXref_i htiter;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.itemtype;
}

uint32_t Terms::getItem(int wterm) {
    htTXref_i htiter;

```

```

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) return 0;
    return htiter->second.item;
}

int Terms::TermLookup(uint32_t wactor, uint32_t waction, uint32_t wappexec,
                      uint32_t witemtype, uint32_t witem) {
    int rc;

    sqlite3_reset(stmt_selRevTrm);
    sqlite3_bind_int(stmt_selRevTrm, 1, wactor);
    sqlite3_bind_int(stmt_selRevTrm, 2, waction);
    sqlite3_bind_int(stmt_selRevTrm, 3, wappexec);
    sqlite3_bind_int(stmt_selRevTrm, 4, witemtype);
    sqlite3_bind_int(stmt_selRevTrm, 5, witem);
    rc = sqlite3_step(stmt_selRevTrm);
    if (rc != SQLITE_ROW) return -1;
    return sqlite3_column_int(stmt_selRevTrm, 0);
}

void Terms::addTerm(int wterm) {
    int rc;
    htTXref_i htiter;
    term_entry txref;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) {
        printf("**addTerm** term %i not found -- not creating anything\n", wterm);
        return;
    }
    txref = htiter->second;

    if (TermLookup(txref.actor, txref.action, txref.appexec, txref.itemtype,
                  txref.item) == -1) {
        sqlite3_reset(stmt_insTrm);
        sqlite3_bind_int(stmt_insTrm, 1, txref.actor);
        sqlite3_bind_int(stmt_insTrm, 2, txref.action);
        sqlite3_bind_int(stmt_insTrm, 3, txref.appexec);
        sqlite3_bind_int(stmt_insTrm, 4, txref.itemtype);
        sqlite3_bind_int(stmt_insTrm, 5, txref.item);
        sqlite3_bind_int(stmt_insTrm, 6, wterm);
        rc = sqlite3_step(stmt_insTrm);
    }
}

```

```

    }
}

int Terms::addStringTerm(int wterm) {
    int rc;
    term_entry txref;

    if (wterm == 0) wterm = maxTerm++;

    txref.action = txref.actor = txref.appexec = txref.item = txref.itemtype = 0;
    txref.isString = 1;
    txref.id = wterm;
    htTXref[wterm] = txref;

    sqlite3_reset(stmt_insTrm);
    sqlite3_bind_int(stmt_insStrTrm, 1, wterm);
    rc = sqlite3_step(stmt_insStrTrm);

    return wterm; // only makes sense if the input is 0
}

int Terms::getNewTermT(int wterm, uint32_t wgeneric, uint8_t allowAdd) {
    htTXref_i htiter;
    term_entry txref;

    if (wterm == -1) return -1;

    htiter = htTXref.find(wterm);
    if (htiter == htTXref.end()) {
        printf("**getNewTermT** can't find term %i \n", wterm);
        return -1;
    }
    else {
        txref = htiter->second;
        return getTermT(txref.actor, txref.action, txref.appexec, txref.itemtype,
                        wgeneric, 0, allowAdd);
    }
}

int Terms::getTermT(uint32_t wactor, uint32_t waction, uint32_t wappexec,
                    uint32_t witemtype, uint32_t witem,
                    uint8_t useGeneric, uint8_t allowAdd) {

```



```

htTerms_i htiter;

uint32_t hashArray[5];
int wterm;
uint32_t whash;
term_entry txref;

if (wactor != 0) wactor = GNRC_ACTOR_INSTANCE;

if (parameters->debugParms >= 6)
    fprintf(parameters->logFile, "**wgetTermT actor = %i\t\t action = %i\t\t app = %i\t\t\n",
hashArray[0] = wactor;
hashArray[1] = waction;
hashArray[2] = wappexec;
hashArray[3] = witemtype;
if (useGeneric == 0)
    hashArray[4] = witem;
else {
    if (witem != 0) {
        hashArray[4] = patterns->getGenericMatch(witem, NULL, allowAdd);
    }
    else {
        hashArray[4] = patterns->getGenericMatch((char *)NULL, NULL, allowAdd);
    }
    if (hashArray[4] == 0) return -1;
}

whash = utilities->hashword((const uint32_t *)hashArray, 5, 0xdeadbeef);
htiter = htTerms.find(whash);
if ((htiter == htTerms.end()) && allowAdd) {
    wterm = maxTerm++;
    htTerms[whash] = wterm;
    parameters->termCount++;
    if (1) printf("\t\t\t\t\tadded Term %i - actor %u action %u application %u itemtype %u\n",
        wterm, wactor, waction, wappexec, witemtype, hashArray[4]);
    txref.action = waction;
    txref.actor = wactor;
    txref.appexec = wappexec;
    txref.item = hashArray[4];
    txref.itemtype = witemtype;
    txref.id = wterm;
    htTXref[wterm] = txref;
}

```

```

        addTerm(wterm);
        return wterm;
    }
    else {
        if (htiter == htTerms.end()) {
            return -1;
        }
    }
    return htiter->second;
}

void Terms::loadTerms() {
    int rc;
    term_entry txref;
    int wterm;
    uint32_t whash, hashArray[5];

    printf("Loading Terms\n");
    sqlite3_reset(stmt_selAllTrm);
    while ((rc = sqlite3_step(stmt_selAllTrm)) == SQLITE_ROW) {
        txref.id = sqlite3_column_int(stmt_selAllTrm, 0);
        txref.actor = sqlite3_column_int(stmt_selAllTrm, 1);
        txref.action = sqlite3_column_int(stmt_selAllTrm, 2);
        txref.appexec = sqlite3_column_int(stmt_selAllTrm, 3);
        txref.itemtype = sqlite3_column_int(stmt_selAllTrm, 4);
        txref.item = sqlite3_column_int(stmt_selAllTrm, 5);
        txref.isString = sqlite3_column_int(stmt_selAllTrm, 6);
        wterm = txref.id;
        htTXref[wterm] = txref;

        if (wterm >= maxTerm) maxTerm = wterm + 1;

        hashArray[0] = txref.actor;
        hashArray[1] = txref.action;
        hashArray[2] = txref.appexec;
        hashArray[3] = txref.itemtype;
        hashArray[4] = txref.item;
        whash = utilities->hashword((const uint32_t *)hashArray, 5, 0xdeadbeef);
        htTerms[whash] = wterm;
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile,
                "Adding to Term hash tables... hash %u\t\tterm %i: actor %u action %u\n",
                whash, txref.id, txref.actor, txref.action, txref.appexec, txref.item);
    }
}

```

}

}

6.6 Strings

Header.

```
#pragma once
#include "lists.h"
#include "queues.h"
#include "TearCPP.h"
#include "Parameters.h"
#include "Terms.h"
#include "InstanceGenericXref.h"
#include "GenericXref.h"
#include "Generics.h"
#include "TermList.h"
#include "Patterns.h"
#include "utilities.h"
#include "ItemUtilities.h"
#include "ItemInstance.h"
#include <unordered_map>

#define htSXref_i std::unordered_map<uint32_t, uint32_t>::const_iterator
#define htStrings_i std::unordered_map<uint32_t, string_entry>::const_iterator
class Strings{
    typedef struct string_type {
        uint32_t id;
        uint32_t length;
        uint32_t totalLength;
        int *terms;
        uint8_t *termTypes;
        uint8_t *wildcards;
        listT<uint32_t> *genericItems;
        char *stringName;
        uint8_t primaryKey;
        uint8_t inDB;
    } string_entry;

    static bool intLessThan(const uint32_t *lhs, const uint32_t *rhs) {
        if (*lhs < *rhs) { return true; }
        if (*lhs > *rhs) { return false; }
        return false; // false if elements are equal
    }

    std::unordered_map<uint32_t, string_entry> htStrings;
```

```

std::unordered_map<uint32_t, uint32_t> htSXref;
ItemUtilities *itemUtilities;
Utilities *utilities;
Parameters *parameters;
Terms *terms;
Generics *generics;

sqlite3_stmt *stmt_insStringTerms;
sqlite3_stmt *stmt_insString;
sqlite3_stmt *stmt_selStringTerms;
sqlite3_stmt *stmt_selString;
sqlite3_stmt *stmt_selStringSupportXref;
sqlite3_stmt *stmt_selSupport;
sqlite3_stmt *stmt_insStrTrm;

public:
    Strings(Utilities *wutilities, ItemUtilities *witemUtilities,
            Parameters *wparameters, Terms *wterms, Generics *wgenerics);
    ~Strings();
    void loadStrings();
    uint32_t getTotalLength(uint32_t wstringid);
    uint32_t getString(int *stringOrder, uint32_t *sourceList,
                      uint32_t listLength, uint32_t primaryID);
    uint32_t getString(TermList *wterm);
    listT<uint32_t> *getAll();
    char *getName(uint32_t wstringid);
    uint8_t getPrimaryKey(uint32_t wstringid);
    uint32_t getLength(uint32_t wstringid);
    uint32_t createString(TermList *wtermList, uint32_t listLength, uint32_t primaryID
    TermList *calcString(TermList *termList, queueT<ItemInstance> *qItems,
                        uint32_t wlength, uint8_t allowAdd, listT<GenericXref> *genXref,
                        listT<InstanceGenericXref> *headInstanceXref, Patterns *wpattern);
    int getStringGeneric(int wterm, listT<GenericXref> *genXref, ItemInstance *witems,
                        listT<InstanceGenericXref> *headInstanceXref, char *orderId, uint8_t allowAdd);
    TermList *getStringTerms(int termid, TermList *wend, char *worder);
    TermList *flatten(TermList *oldTerms, uint32_t *newLength);
    void WriteStrings();
    int getTerm(uint32_t sid, uint32_t windex);
    uint8_t getType(uint32_t sid, uint32_t windex);
    uint8_t getWildcard(uint32_t sid, uint32_t windex);
};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Class: Strings
// Notes:
// 1. Strings are collections of terms that are used to abstract a
// collection of events into a more concise form.
// 2. Strings take the form of  $S = T_1 + T_2 + \dots + T_n$  where  $T_x$  is a
// term. Strings can then be matched to sequences of events to
// replace the sequence with a single entry (String Instance).
// 3. Strings correspond loosely to a regular grammar. For instance,
// the string  $S = T_1 + T_2 + T_3$  corresponds to  $\{S_1 = T_3, S_2 = T_2 + S_1,$ 
//  $S_3 = T_1 + S_2, S = S_3\}$ 
// 4. The advantage of strings over a strict regular grammar is that
// the longer length of strings allows for more fault tolerance to
// noise and interleaving.
// 5. There are four DB files used to maintain traceability.
// StringInstances links the specific string instance (stored as an
// activity in the event table) with the string id used. StringXref
// links the lower level timelines that make up the specific string
// instance. StringSupport links the string instance and the
// generics used with the specific values for them. Finally
// StringSupportXref links the changed variable values when creating
// "strings of strings" (e.g., combining CMAH(X) where  $X = \text{item1}$  and
// CMAH(X) where  $X = \text{item2}$  into CMAH(X) + CMAH(Y) where the new X
// equals the old first X and the new Y equal the old second X.
// Initial Release: 29-Sep-17
//
// Revision History:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "Strings.h"

Strings::Strings(Utilities *wutilities, ItemUtilities *witemUtilities,
                 Parameters *wparameters, Terms *wterms, Generics *wgenerics) {
    char exec_string[2000];
    int rc;

    // Initialize Variables
    itemUtilities = witemUtilities;
    utilities = wutilities;

```

```

parameters = wparameters;
terms = wterms;
generics = wgenerics;

// SQL Statements
sprintf_s(exec_string, 2000,
          "SELECT id, stringname, primarykey, size FROM Strings;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selString, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select String Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000,
          "SELECT term, type, wildcard FROM stringTerms where id = ?1;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000,
          &stmt_selStringTerms, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select String Terms Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000,
          "INSERT INTO strings (id, primaryKey, stringName, size) VALUES (?1, ?2, ' ', ?3);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000,
          &stmt_insString, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Strings Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000,
          "INSERT INTO stringTerms (id, position, term, type, wildcard) VALUES (?1, ?2, ?3, ?4, ?5);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000,
          &stmt_insStringTerms, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert String Terms Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000,
          "INSERT INTO terms (actor, action, appexec, itemtype, item, id, isString) VALUES (?1, ?2, ?3, ?4, ?5, ?6, ?7);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000,

```

```

        &(stmt_insStrTrm), 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare Insert String Terms Statement\n", rc);
        return;
    }
}

Strings::~Strings() {}

uint32_t Strings::getTotalLength(uint32_t wstringid) {
    // Length of current string and all substrings
    htStrings_i htiter;

    htiter = htStrings.find(wstringid);
    if (htiter != htStrings.end()) return htiter->second.totalLength;
    return 0;
}

char *Strings::getName(uint32_t wstringid) {
    htStrings_i htiter;
    // String name

    htiter = htStrings.find(wstringid);
    if (htiter != htStrings.end()) return htiter->second.stringName;
    return NULL;
}

uint8_t Strings::getPrimaryKey(uint32_t wstringid) {
    htStrings_i htiter;

    htiter = htStrings.find(wstringid);
    if (htiter != htStrings.end()) return htiter->second.primaryKey;
    return 0;
}

uint32_t Strings::getLength(uint32_t wstringid) {
    htStrings_i htiter;
    // # of terms in string

    htiter = htStrings.find(wstringid);
    if (htiter != htStrings.end()) return htiter->second.length;
}

```



```

        return 0;
    }

    int Strings::getTerm(uint32_t wstringid, uint32_t windex) {
        htStrings_i htiter;

        htiter = htStrings.find(wstringid);
        if (htiter != htStrings.end()) {
            if (windex < htiter->second.length) return htiter->second.terms[windex];
        }
        return 0;
    }

    uint8_t Strings::getType(uint32_t sid, uint32_t windex) {
        htStrings_i htiter;
        // Get type of windex term

        htiter = htStrings.find(sid);
        if (windex < htiter->second.length) return htiter->second.termTypes[windex];
        return 0;
    }

    uint8_t Strings::getWildcard(uint32_t sid, uint32_t windex) {
        htStrings_i htiter;
        // get wildcard (*, +, nothing) for windex term

        htiter = htStrings.find(sid);
        if (htiter != htStrings.end()) return htiter->second.length;
        return 0;
        if (windex < htiter->second.length) return htiter->second.wildcards[windex];
    }

    uint32_t Strings::getString(TermList *wterm) {
        // Purpose: Return the string id that matches the term list

        int *wterms;
        uint32_t *wsources;
        uint32_t wlength;
        TermList *wtermList;

```

```

uint32_t retVal;

////////////////////////////////////

// Initialize variables
wterms = (int *)malloc(sizeof(int) * parameters->maxString);
wsource = (uint32_t *)malloc(sizeof(uint32_t) * parameters->maxString);

// Build the termlist array
wtermList = wterm;
wlength = 0;
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "GetString -- looking for terms: ");
while (wtermList != NULL) {
    /* commented out to work on Windows 10
    if ((uint64_t)wtermList > 0x7fffffff && (uint64_t)wtermList > 0) {
        if (parameters->debugParms >= 5) fprintf(parameters->logFile, "%i -- returni
        return 0;
    }
    */
    if (wtermList->getTerm() <= 0 || wtermList->getTerm() >= terms->getMaxTerm()) {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "%i -- returning", wtermList->getTerm());
        return 0;
    }
    wterms[wlength] = wtermList->getTerm();
    wsource[wlength] = wtermList->getSource();
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "%i\t",wtermList->getTerm());
    wlength++;
    wtermList = wtermList->getNext();
}
if (parameters->debugParms >= 5) fprintf(parameters->logFile, "----- ");

retVal = getString(wterms, wsource, wlength, 0);
free(wterms);
free(wsource);
if (parameters->debugParms >= 5) fprintf(parameters->logFile, "%i\n",retVal);
return retVal;
}

uint32_t Strings::getString(int *stringOrder, uint32_t *sourceList,
    uint32_t listLength, uint32_t primaryID) {

```

```

// Purpose: return the string id that matches the term array

htSXref_i htiter;
htStrings_i htstring;

uint32_t *hashArray;
uint32_t whash;
uint8_t isMatch;
uint32_t i;
string_entry sentry;

////////////////////////////////////

// Initialize the hash array and then get the hash for the term array
hashArray = (uint32_t *)malloc(sizeof(uint32_t) * listLength);
for (i = 0; i < listLength; i++) {
    hashArray[i] = (stringOrder)[i];
}
whash = utilities->hashword(
    (const uint32_t *)hashArray, listLength, 0xdeadbeef);

// Handle collisions
while (1) {
    htiter = htSXref.find(whash);
    if (htiter == htSXref.end()) {
        free(hashArray);
        return 0;
    }
    htstring = htStrings.find(htiter->second);
    if (htstring == htStrings.end()) {
        free(hashArray);
        return 0;
    }
    sentry = htstring->second;
    isMatch = 1;
    if (listLength == sentry.length) {
        for (i = 0; i < listLength; i++) {
            if (stringOrder[i] != sentry.terms[i] ||
                sourceList[i] != sentry.termTypes[i]) {
                isMatch = 0;
                break;
            }
        }
    }
}

```

```

        if (isMatch) {
            free(hashArray);
            return htiter->second;
        }
    }
    (whash)++;
}
}

```

```

uint32_t Strings::createString(TermList *wtermList,
                               uint32_t listLength, uint32_t primaryID) {
    // CreateString occurs after all other processing, so it's safe
    // to assume that all the terms that will be created, have been
    // created in getTermT (i.e., in the hash table, but not in the
    // database); maxTerm is correct and we can create a record in
    // the term database file

    // assumes all terms are already added to the database

    uint32_t i;

    int rc, sid;
    TermList *wterm;

    //////////////////////////////////////

    // Check if terms are valid
    wterm = wtermList;
    while (wterm != NULL) {
        // if ((uint64_t)wterm > 0x7fffffff && (uint64_t)wterm > 0) return 0;
        if (wterm->getTerm() <= 0 || wterm->getTerm() >= terms->getMaxTerm())
            return 0;
        wterm = wterm->getNext();
    }

    sid = getString(wtermList);
    if (sid != 0) return sid;

    if (primaryID == 0)
        sid = terms->addStringTerm(0);
    else
        sid = primaryID;
}

```

```

sqlite3_reset(stmt_insString);
sqlite3_bind_int(stmt_insString, 1, sid);
sqlite3_bind_int(stmt_insString, 2, (primaryID == 0 ? 1 : 0));
sqlite3_bind_int(stmt_insString, 3, listLength);
rc = sqlite3_step(stmt_insString);

sqlite3_reset(stmt_insStrTrm);
sqlite3_bind_int(stmt_insStrTrm, 1, sid);
sqlite3_step(stmt_insStrTrm);

if ((rc != SQLITE_DONE) && (rc != SQLITE_OK)) {
    printf("Unable to add String   rc = %i (sid = %i, primaryid = %u)\n",
           rc, sid, primaryID);
    return 0;
}

wterm = wtermList;
for (i = 0; i < listLength; i++) {
    if (wterm != NULL) {
        sqlite3_reset(stmt_insStringTerms);
        sqlite3_bind_int(stmt_insStringTerms, 1, sid);
        sqlite3_bind_int(stmt_insStringTerms, 2, i);
        sqlite3_bind_int(stmt_insStringTerms, 3, wterm->getTerm());
        sqlite3_bind_int(stmt_insStringTerms, 4, wterm->getSource());
        sqlite3_bind_int(stmt_insStringTerms, 5, 0);

        rc = sqlite3_step(stmt_insStringTerms);
        if ((rc != SQLITE_DONE) && (rc != SQLITE_OK)) {
            printf("Unable to add String Term rc = %i (sid = %i, primaryid = %u term\n",
                   rc, sid, primaryID, wterm->getTerm());
            return 0;
        }
    }
    wterm = wterm->getNext();
}

if (parameters->debugParms)
    fprintf(parameters->logFile, "**CreateString -- sid = %i\n", sid);
return sid;
}

TermList *Strings::calcString(TermList *termList, queueT<ItemInstance> *qItems,
    uint32_t wlength, uint8_t allowAdd,

```

```

listT<GenericXref> *genXref,
listT<InstanceGenericXref> *headInstanceXref,
Patterns *wpattern) {

//   for each term (i = 1 to # terms)
//       check if item already found (including templates) -- if so, assign
//       item[i] = generic item found
//       -- if not, get another generic item and create a generic/item
//       xref, assign item[i] = new generic
//       newterm[i] = term for that generic item
//   return newterm

ItemInstance *witem;
TermList *oldTerm;
TermList *newTerm, *termPtr;
TermList *newList, *newTail;
uint32_t wgeneric;
uint32_t i;

void *qiter;
ItemInstance *witems;
listT<uint32_t> *genericsUsed;

GenericXref *newXref;
void *ggenXrefiter;

char *orderId;

////////////////////////////////////

i = 0;

genericsUsed = new listT<uint32_t>;
newList = NULL;
newTail = NULL;
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "entering calcstring -- wlength = %i\n", wlength);
// For each term
oldTerm = termList;
qiter = qItems->initializeIterator();
for (i = 0; i < wlength; i++) {

```

```

// boundary check
if (oldTerm->getTerm() <= 0 || oldTerm->getTerm() > terms->getMaxTerm()) {
    printf("**calcString -- term out of range -- returning NULL\n");
    if (newList != NULL) delete newList;
    // pointers in generic used remain valid (they point to things in Patterns
    delete genericsUsed;
    return NULL;
}

witem = qItems->qvalue(qiter);
if (parameters->debugParms >= 4)
    fprintf(parameters->logFile,
        "**Calcstring -- term = %i    source = %i    generic type = %i    first item
        oldTerm->getTerm(), oldTerm->getSource(),
        witem->getGenericType(), witem->getItem());

// Case 1: we need to match a single generic item for this term
if (oldTerm->getSource() != S_LOOP && witem->getGenericType() == GNRC_ITEM) {
    // we need to match a single generic item for this term
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "Matching single generic item\n");

    wgeneric = 0;
    for (ggenXrefiter = genXref->getFront(); ggenXrefiter != NULL;
        ggenXrefiter = genXref->revIterate(ggenXrefiter)) {
        if (itemUtilities->sameAs(genXref->lvalue(
            ggenXrefiter->getItem(), witem->getItem())) {
            wgeneric = genXref->lvalue(ggenXrefiter->getGeneric());
            break;
        }
    }
}

// If we didn't find, go through the list of patterns and take
// the first (longest) match
if (wgeneric == 0) {
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile,
            "Calcstring -- checking patterns (source = %u    generic type = %u)\n
            oldTerm->getSource(), witem->getGenericType());

    wgeneric = wpattern->getGenericMatch(witem->getItem(),
        &genericsUsed, allowAdd);
}

```

```

        // If you couldn't find any patterns (or free generics for
        // those patterns), return NULL
        if (wgeneric == 0) {
            if (newList != NULL) delete newList;
            // pointers in generic used remain valid (they point to
            // things in Patterns
            delete genericsUsed;
            return NULL;
        }

        // Otherwise, add the new cross reference
        newXref = new GenericXref(wgeneric, witem->getItem());
        genXref->insertFront(newXref);
    }

    // Case 2: we need to match loop of items as a single generic item
    //           for this term
}
else if (oldTerm->getSource() == S_LOOP) {
    if (witem->getListID() == 0)
        printf("**calcstring -- source = loop, but list id is zero!\n");
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "Matching loop items\n");

    wgeneric = 0;
    for (ggenXrefiter = genXref->getFront(); ggenXrefiter != NULL;
         ggenXrefiter = genXref->revIterate(ggenXrefiter)) {
        if ((genXref->lvalue(ggenXrefiter)->getLoopList() != NULL) &&
            witem->matchLoopLists(itemUtilities, genXref->lvalue(
                ggenXrefiter)->getLoopList())) {
            wgeneric = genXref->lvalue(ggenXrefiter)->getGeneric();
            break;
        }
    }
}

// If we didn't find, go through the list of patterns and take the
// first (longest) match
if (wgeneric == 0) {
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile,
            "Calcstring -- checking patterns (source = %u   generic type = %u)\n",
            oldTerm->getSource(), witem->getGenericType());
}

```



```

        // NULL automatically uses a star pattern
        wgeneric = wpattern->getGenericMatch((char *)NULL,
            &genericsUsed, allowAdd);

        // If you couldn't find any patterns (or free generics for
        // those patterns), return NULL
        if (wgeneric == 0) {
            if (newList != NULL) delete newList;
            // pointers in generic used remain valid (they point to
            // things in Patterns
            delete genericsUsed;
            return NULL;
        }

        // Otherwise, add the new cross reference
        newXref = new GenericXref(wgeneric, witem->getLoopList(),
            witem->getListID());
        genXref->insertFront(newXref);
    }

    // Case 3: we need to match a list of generic items for this
    //          term (e.g., string with multiple generics)
}
else { // genericType == GNRC_ITEMLIST && !LOOP
//     printf("*****Case3\n");
    if (witem->getListID() != 0)
        printf("**calcstring -- source != loop, but list id is not zero!    (source\n",
            oldTerm->getSource(), oldTerm->getTerm(),
            witem->getGenericType(), witem->getItem(),
            (uint64_t)(witem->getItemList()));
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "Matching string generics\n");

    // Match each item in the generic list
    witems = witem->getItemList();
    while (witems != NULL) {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "\ttrying to match %i    \n",
                witems->getItem());

        wgeneric = 0;
        for (ggenXrefiter = genXref->getFront(); ggenXrefiter != NULL;
            ggenXrefiter = genXref->revIterate(ggenXrefiter)) {

```

```

        if (itemUtilities->sameAs(genXref->lvalue(
            ggenXrefiter)->getItem(), witems->getItem())) {
            wgeneric = genXref->lvalue(ggenXrefiter)->getGeneric();
            break;
        }
    }

    // If we didn't find one, go through the list of patterns and
    // take the first (longest) match
    if (wgeneric == 0) {
        if (parameters->debugParms >= 4)
            fprintf(parameters->logFile,
                "Calcstring -- checking patterns (source = %u    generic type = %u\n",
                oldTerm->getSource(), witems->getGenericType());

        wgeneric = wpattern->getGenericMatch(witems->getItem(),
            &genericsUsed, allowAdd);

        // If you couldn't find any patterns (or free generics for
        // those patterns), return NULL
        if (wgeneric == 0) {
            if (newList != NULL) delete newList;
            // pointers in generic used remain valid (they point to
            // things in Patterns
            delete genericsUsed;
            return NULL;
        }

        newXref = new GenericXref(wgeneric, witems->getItem());
        genXref->insertFront(newXref);
    }

    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile,
            "***calcString**    item = %u (%u)    generic = %u\n",
            witems->getItem(), witems->getGenericType(), wgeneric);
    witems = witems->getNext();
}

// Using the original actor, action, app, and itemtype and the
// (possibly) new generic, get the term
if (parameters->debugParms >= 4)

```

```

        fprintf(parameters->logFile, "**calcstring**      term = %i      source = %u\n",
            oldTerm->getTerm(), oldTerm->getSource());
    if (oldTerm->getSource() != S_STRING) {
        newTerm = new TermList(0, oldTerm->getSource(), oldTerm->getOrder(),
            oldTerm->getOrderid(), NULL);
        newTerm->setTerm((allowAdd ? terms->getNewTermA(oldTerm->getTerm(),
            wgeneric) : terms->getNewTerm(oldTerm->getTerm(), wgeneric)));
    }
    else {
        orderId = (char *)malloc(sizeof(char) * (3 + 2));
        if (orderId == NULL) {
            printf("**calcString-- unable to allocate orderid - returning\n");
            if (newList != NULL) delete newList;
            // pointers in generic used remain valid (they point to things
            // in Patterns
            delete genericsUsed;
            return NULL;
        }
        sprintf_s(orderId, 3 + 1, "%u", i);
        newTerm = new TermList(0, oldTerm->getSource(), oldTerm->getOrder(),
            oldTerm->getOrderid(), NULL);
        newTerm->setTerm(getStringGeneric(oldTerm->getTerm(), genXref,
            witem, headInstanceXref, orderId, allowAdd));
        free(orderId);

        if (newTerm->getTerm() == -1) {
            if (parameters->debugParms >= 4)
                fprintf(parameters->logFile,
                    "couldn't find string for term %i -- returning\n",
                    oldTerm->getTerm());
            delete newTerm;
            if (newList != NULL) delete newList;
            // pointers in generic used remain valid (they point to things
            // in Patterns
            delete genericsUsed;
            return NULL;
        }
    }

}

if (newList == NULL) newList = newTerm;
if (newTail != NULL) newTail->setNext(newTerm);
newTail = newTerm;

```

```

        oldTerm = oldTerm->getNext();
        qiter = qlItems->iterate(qiter);
    }

    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "calcstring...leaving\n");

    delete genericsUsed; // pointers in generic used remain valid (they point to
                          // things in Patterns

    if (parameters->debugParms >= 4) {
        fprintf(parameters->logFile, "Returning: \t");
        for (termPtr = newList; termPtr != NULL; termPtr = termPtr->getNext()) {
            fprintf(parameters->logFile, "%i\t", termPtr->getTerm());
        }
        fprintf(parameters->logFile, "\n");
    }
    return newList;
}

int Strings::getStringGeneric(int wterm, listT<GenericXref> *genXref,
                               ItemInstance *witems,
                               listT<InstanceGenericXref> *headInstanceXref,
                               char *orderId, uint8_t allowAdd) {

    // Purpose: fix string-of-strings issue by syncing generics across strings

    htStrings_i htstring;
    htSXref_i htiter;

    listT<GenericXref> *oldxref;
    void *liter;
    ItemInstance *xitems, *witemList;
    string_entry sxref, sxref2;
    GenericXref *genXrefptr, *tmpGenXref;
    uint32_t stringGenericItem;
    uint32_t i;
    int *xterm;
    char *newOrderId;
    uint32_t *witemid;

```

```

int wgenericItem;
uint32_t wstring, whash, *hashArray, witem, lastitem;
InstanceGenericXref *winstanceXref;

void *ggenXrefiter;

////////////////////////////////////

oldxref = new listT<GenericXref>;

if (parameters->debugParms >= 4)
    fprintf(parameters->logFile, "**getStringGeneric**   string = %i\n", wterm);

// First look up the string
htstring = htStrings.find(wterm);
if (htstring == htStrings.end()) {
    printf("**Error -- getStringGeneric -- couldn't find string %i -- returning\n"
    delete oldxref;
    return -1;
}
sxref = htstring->second;

// Create a table of the cross reference between the items for this
// particular instance and the generics for this particular string.
// if the generics for this string are not available, start with
// generic item 0

// only one generic (could be a generic for a loop)
if (witems->getGenericType() == GNRC_ITEM || witems->getListID() != 0)
    xitems = witems;
else // more than one -- use witems itemlist
    xitems = witems->getItemList();

//
// oldXref:
//      item/ itemlist = item# sent into calcstring
//      generic = the generic# in the master string table
if (xitems != NULL) {
    lastitem = 0;
    for (liter = sxref.genericItems->getFront(); liter != NULL;
        liter = sxref.genericItems->revIterate(liter)) {
        // only want one instance of each item in the cross reference

```

```

        if (lastitem != 0 && itemUtilities->sameAs(lastitem,
            *(sxref.genericItems->lvalue(liter))))
            continue;

        lastitem = *(sxref.genericItems->lvalue(liter));
        genXrefptr = new GenericXref(*(sxref.genericItems->lvalue(liter)),
            xitems->getItem(), xitems->getGenericType(),
            xitems->getItemList(), xitems->getLoopList(),
            xitems->getListID());
        oldxref->insertFront(genXrefptr);

        if (witems->getGenericType() == GNRC_ITEM ||
            witems->getListID() != 0)
            break;

        xitems = xitems->getNext();
        if (xitems == NULL) break;
    }
}

// Show the old and new Xref
if (parameters->debugParms >= 5) {
    fprintf(parameters->logFile, "**getStringGeneric**\n");
    for (ggenXrefiter = genXref->getFront(); ggenXrefiter != NULL;
        ggenXrefiter = genXref->revIterate(ggenXrefiter)) {
        fprintf(parameters->logFile, "\tNEW genXref:   %u -- %u or %llx\n",
            genXref->lvalue(ggenXrefiter)->getGeneric(),
            genXref->lvalue(ggenXrefiter)->getItem(),
            (uint64_t)(genXref->lvalue(ggenXrefiter)->getItemList()));
    }

    for (ggenXrefiter = oldxref->getFront(); ggenXrefiter != NULL;
        ggenXrefiter = oldxref->revIterate(ggenXrefiter)) {
        fprintf(parameters->logFile, "\tOLD genXref:   %u -- %u or %llx\n",
            oldxref->lvalue(ggenXrefiter)->getGeneric(),
            oldxref->lvalue(ggenXrefiter)->getItem(),
            (uint64_t)(oldxref->lvalue(ggenXrefiter)->getItemList()));
    }
}

```

```

// Xterm will hold the new terms
xterm = (int *)malloc(sizeof(int) * sxref.length);

// Now, we'll go through each term in the original string and do the xref
// between the old assignment of generic items and the new one
// e.g., stringsupport says Generic1 = 123 but the GenXref from Calcstring
// says 123 = Generic2 this will link the old generic1 with the new generic2
for (i = 0; i < sxref.length; i++) {
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "getStringGeneric -- term = %i    type = %u\n",
            sxref.terms[i], sxref.termTypes[i]);

    // if this term is a string, continue down the tree
    if (sxref.termTypes[i] == S_STRING) { // if a string, repeat the process
        newOrderId = (char *)malloc(sizeof(char) *
            (strlen(orderId) + 1 + 3 + 2));
        if (newOrderId == NULL) {
            printf("**getStringGeneric -- unable to allocate newOrderId -- returning\n");
            for (liter = oldxref->getFront(); liter != NULL;
                liter = oldxref->revIterate(liter)) {
                tmpGenXref = oldxref->lvalue(liter);
                if (tmpGenXref != NULL) delete tmpGenXref;
            }
            free(xterm);
            delete oldxref;
            return -1;
        }
        sprintf_s(newOrderId, strlen(orderId) + 1 + 3 + 1, "%s.%u",
            orderId, i);
        xterm[i] = getStringGeneric(sxref.terms[i], genXref, witems,
            headInstanceXref, newOrderId, allowAdd);
        free(newOrderId);
        continue;
    }

    // Get the generic item for the current term in the original string
    stringGenericItem = terms->getItem(sxref.terms[i]);

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "getStringGeneric -- looking for item %i\n",

```

```

        stringGenericItem); //535932

// Now go back through the xref for the original string and find the
// actual item/itemlist from the stringsupport file
// figure out what the itemid (or itemlist) for the string's generic
// item is
witem = 0;
witemList = NULL;
for (ggenXrefiter = oldxref->getFront(); ggenXrefiter != NULL;
     ggenXrefiter = oldxref->revIterate(ggenXrefiter)) {
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "oldxref->generic = %i      stringGenericItem = %i\n",
            oldxref->lvalue(ggenXrefiter)->getGeneric(),
            stringGenericItem);
    if (oldxref->lvalue(ggenXrefiter)->getGeneric() ==
        stringGenericItem) {
        if (oldxref->lvalue(ggenXrefiter)->getGenericType() ==
            GNRC_ITEM) {
            witem = oldxref->lvalue(ggenXrefiter)->getItem();
            if (parameters->debugParms >= 5)
                fprintf(parameters->logFile,
                    "\tgetStringGeneric -- looking for item %i\n",
                    witem); //535932

        }
        else {
            witemList = oldxref->lvalue(ggenXrefiter)->getItemList();
            if (parameters->debugParms >= 5)
                fprintf(parameters->logFile,
                    "\tgetStringGeneric -- looking for item list %llx\n",
                    (uint64_t)witemList); //535932
        }
        break;
    }
}

// **now figure out what the (possibly different) generic item is for
// that itemid (or itemlist)
wgenericItem = 0;
for (ggenXrefiter = genXref->getFront(); ggenXrefiter != NULL;
     ggenXrefiter = genXref->revIterate(ggenXrefiter)) {
    if ((genXref->lvalue(ggenXrefiter)->getGenericType() == GNRC_ITEM &&

```



```

        itemUtilities->sameAs(genXref->lvalue(ggenXrefiter)->getItem(), witem)) |
        (genXref->lvalue(ggenXrefiter)->getGenericType() == GNRC_ITEMLIST &&
         witemList != NULL && witemList->matchItemLists(itemUtilities,
         genXref->lvalue(ggenXrefiter)->getItemList()))
    ) {
        wgenericItem = genXref->lvalue(ggenXrefiter)->getGeneric();
        break;
    }
}

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "\t\tstring item = %u    instance item = %u    new generic = %i\n",
        stringGenericItem, witem, wgenericItem);
if (wgenericItem == 0) {
    printf("**Error -- getStringGeneric - couldn't find generic for item %u -- r
    fprintf(parameters->logFile, "**Error -- getStringGeneric - couldn't find gen
    for (liter = oldxref->getFront(); liter != NULL;
        liter = oldxref->revIterate(liter)) {
        tmpGenXref = oldxref->lvalue(liter);
        if (tmpGenXref != NULL) delete tmpGenXref;
    }
    free(xterm);
    delete oldxref;
    return -1;
}

// and then re-lookup it up with that generic item

// Now get the term for the string with the new (i.e., genxref) generic
// item
// note -- this may not exist, so if you have permission, you may have
// to add it
// e.g., CMAH works with generic item 1, but when I combine it with
//      CM(1)CM(2) (e.g., CM(1)CM(2)CMAH(2), I need to create a CMAH(2)
xterm[i] = (allowAdd ? terms->getNewTermA(sxref.terms[i], wgenericItem)
                  : terms->getNewTerm(sxref.terms[i], wgenericItem));
if (xterm[i] == -1) return -1;

// Now create a record of the cross reference between the generic items
if (headInstanceXref != NULL) {
    newOrderId = (char *)malloc(sizeof(char) * (strlen(orderId) + 2));
    strcpy_s(newOrderId, strlen(orderId) + 1, orderId);

```

```

        winstanceXref = new InstanceGenericXref(newOrderId,
            stringGenericItem, wgenericItem);
        free(newOrderId);
        headInstanceXref->insertFront(winstanceXref);
    }

}

for (liter = oldxref->getFront(); liter != NULL;
    liter = oldxref->revIterate(liter)) {
    tmpGenXref = oldxref->lvalue(liter);
    if (tmpGenXref != NULL) delete tmpGenXref;
}
delete oldxref;

// now do a reverse lookup with the hashes for the new terms
hashArray = (uint32_t *)malloc(sizeof(uint32_t) * sxref.length);
if (hashArray == NULL) {
    printf("**getStringGeneric -- error -- unable to allocate hashArray\n");
    return -1;
}
for (i = 0; i < sxref.length; i++) {
    hashArray[i] = xterm[i];
}
whash = utilities->hashword((const uint32_t *)hashArray, sxref.length,
    0xdeadbeef);
free(hashArray);

htiter = htSXref.find(whash);

// if you found the string id, return it
if (htiter != htSXref.end()) {
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile,
            "**getStringGeneric -- returning %i\n", htiter->second);
    free(xterm);
    return htiter->second;
}

if (allowAdd == 0) return -1;

// if not found, create it in the hash tables (it'll be added to the
// DB later) and return it;

```

```

wstring = terms->addStringTerm(0);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "creating string %u\n", wstring);
sxref2.id = wstring;
sxref2.stringName = NULL;
sxref2.primaryKey = sxref.primaryKey;
sxref2.inDB = 0;
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "new length = %u\n", sxref.length);
sxref2.length = sxref.length;
sxref2.totalLength = sxref.totalLength;
sxref2.genericItems = NULL;
sxref2.terms = (int *)malloc(sizeof(int) * sxref2.length);
if (sxref2.terms == NULL) {
    printf("**getString -- unable to allocate terms - returning\n");
    return -1;
}
sxref2.termTypes = (uint8_t *)malloc(sizeof(uint8_t) * sxref2.length);
if (sxref2.termTypes == NULL) {
    printf("**getString -- unable to allocate termTypes - returning\n");
    return -1;
}
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "assigning terms and types\n");
for (i = 0; i < sxref2.length; i++) {
    (sxref2.terms)[i] = xterm[i];
    (sxref2.termTypes)[i] = sxref.termTypes[i];
    witemid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
    *witemid = terms->getItem(xterm[i]);
    if (xterm[i] != 0 && generics->isGenericItem(*witemid)) {
        if (sxref2.genericItems != NULL) {
            sxref.genericItems = new listT<uint32_t>;
            sxref.genericItems->insertSort(witemid, &intLessThan);
        }
        else {
            sxref.genericItems->insertSort(witemid, &intLessThan);
        }
    } else free(witemid);
}
if (parameters->debugParms >= 5) {
    fprintf(parameters->logFile,
        "getString -- Adding to String hash tables... id %u:",
        sxref2.id);
}

```

```

        for (i = 0; i < sxref2.length; i++) {
            fprintf(parameters->logFile, "\t%i", (sxref2.terms)[i]);
        }
        fprintf(parameters->logFile, "\n");
    }
    htStrings[sxref2.id] = sxref2;
    hashArray = (uint32_t *)malloc(sizeof(uint32_t) * sxref2.length);
    if (hashArray == NULL) {
        printf("**Error -- getString - couldn't allocate hashArray - returning\n");
        return -1;
    }
    for (i = 0; i < sxref2.length; i++) {
        hashArray[i] = (sxref2.terms)[i];
    }
    whash = utilities->hashword((const uint32_t *)hashArray,
                               sxref2.length, 0xdeadbeef);
    free(hashArray);
    htSXref[whash] = sxref2.id;

    free(xterm);

    return wstring;
}

listT<uint32_t> *Strings::getAll() {
    // Return a list of strings in the table

    htStrings_i htiter;
    listT<uint32_t> *retList;
    uint32_t *witem;

    retList = new listT<uint32_t>;
    for (htiter = htStrings.begin(); htiter != htStrings.end(); htiter++) {
        witem = (uint32_t *)malloc(sizeof(uint32_t) * 1);
        *witem = htiter->second.id;
        retList->insertRear(witem);
    }
    return retList;
}

void Strings::WriteStrings() {
    htStrings_i htiter;

```

```

string_entry sentry;
int rc;
uint32_t i;
int *wterms;
uint8_t *wsources;

for (htiter = htStrings.begin(); htiter != htStrings.end(); htiter++) {
    sentry = htiter->second;
    if (sentry.inDB == 0) {
        terms->addStringTerm(sentry.id);

        sqlite3_reset(stmt_insString);
        sqlite3_bind_int(stmt_insString, 1, sentry.id);
        sqlite3_bind_int(stmt_insString, 2, 1);
        sqlite3_bind_int(stmt_insString, 3, sentry.length);
        rc = sqlite3_step(stmt_insString);
        if ((rc != SQLITE_DONE) && (rc != SQLITE_OK)) {
            printf("Unable to add String    rc = %i (sid = %u)\n",
                rc, sentry.id);
            continue;
        }

        wterms = sentry.terms;
        wsources = sentry.termTypes;
        for (i = 0; i < sentry.length; i++) {
            sqlite3_reset(stmt_insStringTerms);
            sqlite3_bind_int(stmt_insStringTerms, 1, sentry.id);
            sqlite3_bind_int(stmt_insStringTerms, 2, i);
            sqlite3_bind_int(stmt_insStringTerms, 3, wterms[i]);
            sqlite3_bind_int(stmt_insStringTerms, 4, wsources[i]);
            sqlite3_bind_int(stmt_insStringTerms, 5, 0);
            rc = sqlite3_step(stmt_insStringTerms);
            if ((rc != SQLITE_DONE) && (rc != SQLITE_OK)) {
                printf("Unable to add String Term  rc = %i (sid = %u, term = %i)\n",
                    rc, sentry.id, wterms[i]);
                continue;
            }
        }
    }
}
}
}

```

```

TermList *Strings::getStringTerms(int termid, TermList *wend, char *worder) {
    htStrings_i htiter;

    uint32_t i;
    TermList *retPtr, *wterm, *wprev;
    string_entry wstring;

    retPtr = NULL;
    htiter = htStrings.find(termid);
    if (htiter != htStrings.end()) {
        wstring = htiter->second;
        wprev = NULL;
        wterm = NULL;
        for (i = 0; i < wstring.length; i++) {
            wterm = new TermList(wstring.terms[i],
                                wstring.termTypes[i], i, worder, NULL);
            if (wprev != NULL) wprev->setNext(wterm);
            if (retPtr == NULL) retPtr = wterm;
            wprev = wterm;
        }
        if (wterm != NULL) wterm->setNext(wend);
    }
    if (retPtr != NULL) return retPtr; else return wend;
}

TermList *Strings::flatten(TermList *oldTerms, uint32_t *newLength) {
    // Turns a string-of-strings into a 1-deep list of terms

    TermList *newTerms, *wptr, *wterms, *prev;

    if (oldTerms == NULL) return NULL;

    newTerms = NULL;
    wptr = oldTerms;
    prev = NULL;
    while (wptr != NULL) {
        wterms = new TermList(wptr->getTerm(), wptr->getSource(),
                              wptr->getOrder(), wptr->getOrderid(), NULL);
        if (prev != NULL) prev->setNext(wterms);
        if (newTerms == NULL) newTerms = wterms;
        prev = wterms;
    }
}

```

```

    wptr = wptr->getNext();
}

wterms = newTerms;
prev = NULL;
if (newLength != NULL) *newLength = 0;
while (wterms != NULL) {
    if (terms->isString(wterms->getTerm())) {
        wptr = getStringTerms(wterms->getTerm(),
                               wterms->getNext(), NULL);
        if (prev != NULL) prev->setNext(wptr);
        delete wterms;
        wterms = wptr;
    } else {
        prev = wterms;
        wterms = wterms->getNext();
        if (newLength != NULL) (*newLength)++;
    }
}

return newTerms;
}

void Strings::loadStrings() {
    htStrings_i htstring, htstring2;
    htSXref_i htiter;

    void *liter;
    int rc, rc2;
    string_entry sxref;
    uint32_t whash, *hashArray;
    int wtermid;
    int wstring;
    uint32_t i;

    uint8_t isFirst;
    char *sname;
    uint32_t wlength;
    uint32_t *witemid;

    printf("Loading Strings\n");
    sqlite3_reset(stmt_selString);
    while ((rc = sqlite3_step(stmt_selString)) == SQLITE_ROW) {

```

```

sxref.id = sqlite3_column_int(stmt_selString, 0);
sxref.primaryKey = sqlite3_column_int(stmt_selString, 1);
sname = (char *)sqlite3_column_text(stmt_selString, 2);
if (sname != NULL)
    sxref.stringName = (char *)malloc(sizeof(char) *
                                     (strlen(sname) + 2));
if (sname != NULL)
    strcpy_s(sxref.stringName, strlen(sname) + 1, sname);
sxref.inDB = 1;
sxref.length = sqlite3_column_int(stmt_selString, 3);
sxref.terms = (int *)malloc(sizeof(int) * sxref.length);
if (sxref.terms == NULL) {
    printf("**loadStrings -- unable to allocate terms - returning\n");
    return;
}
sxref.termTypes = (uint8_t *)malloc(sizeof(uint8_t) * sxref.length);
if (sxref.termTypes == NULL) {
    printf("**loadStrings -- unable to allocate termTypes - returning\n");
    return;
}
sxref.wildcards = (uint8_t *)malloc(sizeof(uint8_t) * sxref.length);
if (sxref.wildcards == NULL) {
    printf("**loadStrings -- unable to allocate wildcards - returning\n");
    return;
}
sxref.totalLength = 0;
sxref.genericItems = new listT<uint32_t>;

sqlite3_reset(stmt_selStringTerms);
sqlite3_bind_int(stmt_selStringTerms, 1, sxref.id);
wlength = 0;
while ((rc2 = sqlite3_step(stmt_selStringTerms)) == SQLITE_ROW) {
    wtermid = sqlite3_column_int(stmt_selStringTerms, 0);
    if (wtermid >= 0) {
        witemid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
        *witemid = terms->getItem(wtermid);
        if (generics->isGenericItem(*witemid)) {
            sxref.genericItems->insertSort(witemid, &intLessThan);
        } else free(witemid);

        htstring2 = htStrings.find(wtermid);
        sxref.totalLength += (htstring2 == htStrings.end() ? 1 :
                               htstring2->second.totalLength);
    }
}

```



```

    if (htstring2 != htStrings.end()) {
        for (liter = (htstring2->second.genericItems)->getFront();
            liter != NULL;
            liter = (htstring2->second.genericItems)->revIterate(liter))
            witemid = (htstring2->second.genericItems)->lvalue(liter);
        if (generics->isGenericItem(*witemid)) {
            if (sxref.genericItems == NULL) {
                sxref.genericItems = new listT<uint32_t>;
                sxref.genericItems->insertSort(witemid,
                                                &intLessThan);
            } else
                sxref.genericItems->insertSort(witemid,
                                                &intLessThan);
        }
    }
    // assuming consistent input....
    (sxref.terms)[wlength] =
        sqlite3_column_int(stmt_selStringTerms, 0);
    (sxref.termTypes)[wlength] =
        sqlite3_column_int(stmt_selStringTerms, 1);
    (sxref.wildcards)[wlength] =
        sqlite3_column_int(stmt_selStringTerms, 2);
    wlength++;
}

if (wlength != sxref.length) {
    printf("**loadStrings**  lengths for string id %u do not match (summary = %u\n",
           sxref.id, sxref.length, wlength, wlength);
    sxref.length = wlength;
}

if (parameters->debugParms >= 3) {
    fprintf(parameters->logFile, "Adding to String hash tables... id %u:",
            sxref.id);
    for (i = 0; i < sxref.length; i++) {
        fprintf(parameters->logFile, "\t%i", (sxref.terms)[i]);
    }
    fprintf(parameters->logFile, "\n");
    isFirst = 1;
    for (liter = sxref.genericItems->getFront(); liter != NULL;

```

```

        liter = sxref.genericItems->revIterate(liter)) {
    if (isFirst)
        fprintf(parameters->logFile, "\tgenerics: ");
    else
        fprintf(parameters->logFile, ", ");
    fprintf(parameters->logFile,
            "%u", *(sxref.genericItems->lvalue(liter)));
    isFirst = 0;
}
fprintf(parameters->logFile, "\n");
}

wstring = sxref.id;
htStrings[sxref.id] = sxref;
hashArray = (uint32_t *)malloc(sizeof(uint32_t) * sxref.length);
for (i = 0; i < sxref.length; i++) {
    hashArray[i] = (sxref.terms)[i];
}
whash = utilities->hashword((const uint32_t *)hashArray, sxref.length,
                            0xdeadbeef);
htSXref[whash] = sxref.id;
}
}

```

6.7 ActivityBag

Header.

```
#pragma once
#include <unordered_map>
#include "queues.h"
#include "lists.h"
#include "TearCPP.h"
#include "Utilities.h"
#include "TermList.h"
#include "TimelineEntry.h"
#include "Strings.h"

#define htActivityBag_i std::unordered_map<uint32_t, ActivityBag *>::const_iterator
class Strings;      // need for classes that include each other in the definitions

class ActivityBag
{
    ActivityBag *variantNext; // For DVSM-T, a circular linked list of variants
                             // of abags; they may not all be variants of each other
    ActivityBag *variantPrev;
    ActivityBag *mainActivityBag; // once the circular linked list is resolved
                                 // and a "Main" bag is determined, put it in this
                                 // field for the variants (and null for itself)

    uint8_t isMain;

    TermList *activities;
    uint32_t length;
    uint32_t totalLength;
    uint32_t count;
    uint32_t genericCount;
    uint32_t stringID;

    queueT<TimelineEntry> *timelines;

    int *activityTable;
    uint32_t *sourceTable;
    uint32_t *countTable; // countTable: 2-dimensional: [i][j] = # of times activity
                          //                               i occurs in the jth position
                          //                               (e.g., [i][2]: # of times
                          //                               activity i is 2nd in list)
    uint8_t removeIt;
```

```

public:
    ActivityBag();
    uint32_t ActivityBag::addActivityBag(queueT<TimelineEntry> *head,
        uint32_t *orderList, uint32_t wlength,
        uint8_t add_tbags, uint8_t add_tables,
        Strings *strings,
        std::unordered_map<uint32_t, ActivityBag *> *htActivityBags,
        Parameters *parameters, Utilities *utilities,
        Patterns *wpattern);
    uint8_t createString(std::unordered_map<uint32_t, ActivityBag *> *htActivityBags,
        Strings *strings);
    uint32_t getCount() { return count; }
    uint32_t getLength() { return length; }
    uint32_t getTotalLength() { return totalLength; }
    uint32_t getGenericCount() { return genericCount; }
    uint32_t getIsMain() { return isMain; }
    TermList *getActivities() { return activities; }
    ActivityBag *getMainActivityBag() { return mainActivityBag; }
    ActivityBag *getVariantNext() { return variantNext; }
    ActivityBag *getVariantPrev() { return variantPrev; }
    uint32_t *getCountTable() { return countTable; }
    uint32_t *getSourceTable() { return sourceTable; }
    uint8_t getRemove() { return removeIt; }
    int *getActivityTable() { return activityTable; }
    uint32_t getStringID() { return stringID; }
    queueT<TimelineEntry> *getTimelines() { return timelines; }
    uint8_t setVariantNext(ActivityBag *other) { variantNext = other; return 1; }
    uint8_t setVariantPrev(ActivityBag *other) { variantPrev = other; return 1; }
    uint8_t setMainActivityBag(ActivityBag *other) { mainActivityBag = other; return 1; }
    uint8_t setIsMain(uint8_t wismain) { isMain = wismain; return 1; }
    uint8_t setActivities(TermList *wterm) { activities = wterm; return 1; }
    uint8_t setStringID(uint32_t wstringid) { stringID = wstringid; return 1; }
    void setRemove() { removeIt = 1; }
    void checkVariations(std::unordered_map<uint32_t, ActivityBag *> *htActivityBags,
        float threshold, Utilities *utilities);
    ~ActivityBag();
};

```

C++.

```

////////////////////////////////////
//                                                                    //
// Class: ActivityBag                                                //
// Input: a queue of terms to convert into an activitybag (stored in a passed//

```

```

//      unordered_map)
// Output: n/a
// Notes:
//      1. This is a class with a single method -- create an activity bag and
//      stored it in the passed unordered map. The importance of it is
//      that it's an intermediary between the production rule generation
//      algorithms and calcstring (called directly from the string
//      abstraction class.
// Initial Release: 29-Sep-17
//
// Revision History:
//
////////////////////////////////////
#include "ActivityBag.h"

ActivityBag::ActivityBag() :
    variantNext(NULL), variantPrev(NULL), mainActivityBag(NULL), activities(NULL),
    activityTable(NULL), countTable(NULL), sourceTable(NULL),
    length(0), isMain(0), totalLength(0), count(0), genericCount(0),
    stringID(0), removeIt(0)
{timelines = new queueT<TimelineEntry>;}

ActivityBag::~ActivityBag()
{
    if (activities != NULL) delete activities;
    if (timelines != NULL) timelines->qclear();
    if (activityTable != NULL) free(activityTable);
    if (sourceTable != NULL) free(sourceTable);
    if (countTable != NULL) free(countTable);
}

uint32_t ActivityBag::addActivityBag(
    queueT<TimelineEntry> *qhead, // entries that contain terms for new string
    uint32_t *orderList,         // the order of the entries
    uint32_t wlength,            // the number of entries in the queue to consider
                                // (which may NOT correspond to # entries in queue
    uint8_t add_tbags,           // true if we need to associate timelines w/ this aBag
    uint8_t add_tables,          // true if we need track ordering of terms in table for
    Strings *strings,            // gives access to the calcString routine (among others)
    std::unordered_map<uint32_t, ActivityBag *> *htActivityBags,
                                // the unordered map that holds all the aBags
    Parameters *parameters,      // gives access to the parameters and general methods
    Utilities *utilities,

```

```

Patterns *patterns) {

// if we're keeping track of timelines, this points to the first one
// note: since we know how many we need (wlength) and there is only one order,
// we only need to keep the first
TimelineEntry *wtimelines;

TermList *newTerms;          // the terms passed back from calcstring
ActivityBag *abag;           // the activity bag created or updated
uint32_t stringLength;       // Used to calculate the total length of the string
uint32_t windex;             // Used to update the counts for the terms in the tables

// Used to count the number of generics in the string passed back from calcString
listT<GenericXref> *genXref;
GenericXref *tmpGenXref;

// Used to build the terms and items to pass to calcString
TermList *wtermList, *wterm, *wprevTerm;
ItemInstance *witem;
TimelineEntry *wentry;
queueT<ItemInstance> *qItems;

// Used to confirm there is only one actor per string
uint32_t wactor, actorInstance;

// Used to deal with collisions
int *stringOrder, *stringTest;
uint32_t whash;
uint8_t match;

//Iterators and counters
htActivityBag_i htiter;
void *qiter, *qitemIter;
void *liter;
uint32_t i;

////////////////////////////////////

// Ensure all generic actors are the same
wactor = 0;
i = 0;
for (qiter = qhead->initializeIterator(); qiter != NULL;

```

```

        qiter = qhead->iterate(qiter)) {
    if (i++ == wlength) break;
    actorInstance = qhead->qvalue(qiter)->getActor();
    if (actorInstance == 0) continue; // N/A is only exception to 1 actor/ string
    if (wactor == 0) wactor = actorInstance;
    else if (actorInstance != wactor) return 0;
}

// boundary check
qiter = qhead->initializeIterator();
if (qiter == NULL) return 0;

// Initialize Variables
wtimelines = (add_tbags ? qhead->qvalue(qiter) : NULL);

// First create a list of terms and items based on the timeline entries that
// were passed
wprevTerm = NULL;
wtermlist = NULL;
qItems = new queueT<ItemInstance>;
for (i = 0; i < wlength; i++) {
    wentry = qhead->qvalue(qiter);

    wterm = wentry->getTermList();
    wterm->setNext(NULL);
    wterm->setOrder(i);
    if (wprevTerm != NULL) wprevTerm->setNext(wterm);
    if (wtermlist == NULL) wtermlist = wterm;
    wprevTerm = wterm;

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "\tactivity = %i\n", wentry->getActivity());

    qItems->qpush(wentry->getItems());
    qiter = qhead->iterate(qiter);
    if (qiter == NULL) break; // should never happen
}

// If we're debugging, show the passed in values
if (parameters->debugParms >= 5) {
    fprintf(parameters->logFile, "**addAbag - terms & items\n");
    wterm = wtermlist;
    qitemIter = qItems->initializeIterator();

```

```

    for (i = 0; i < wlength; i++) {
        witem = qItems->qvalue(qitemIter);
        fprintf(parameters->logFile,
            "\tterm = %i    source = %i    item = %i    generic type = %i\n",
            wterm->getTerm(), wterm->getSource(), witem->getItem(),
            witem->getGenericType());
        wterm = wterm->getNext();
        qitemIter = qItems->iterate(qitemIter);
    }
}

// Calculate a string based on the list of terms
genXref = new listT<GenericXref>;
newTerms = strings->calcString(wtermlist, qItems, wlength, 1, genXref,
    NULL, patterns);

// If we couldn't find a string, clean up what you started and return
if (newTerms == NULL) {
    delete qItems;
    for (liter = genXref->getFront(); liter != NULL;
        liter = genXref->revIterate(liter)) {
        tmpGenXref = genXref->lvalue(liter);
        if (tmpGenXref != NULL) delete tmpGenXref;
    }
    delete genXref;
    return 0;
}

// Hash the terms and look for a match
// Note: since you're only looking at abags that collide with whash,
//       odds of finding a candidate are slight
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "**addAbag - back from calcstring\n");
stringOrder = (int *)malloc(sizeof(int) * wlength);
stringTest = (int *)malloc(sizeof(int) * wlength);
wterm = newTerms;
for (i = 0; i < wlength; i++) stringOrder[i] = 0;
for (i = 0; i < wlength; i++) {
    if (wterm == NULL) break;
    stringOrder[i] = wterm->getTerm();
    if (parameters->debugParms >= 3)
        fprintf(parameters->logFile, "\tTerm = %i\n", stringOrder[i]);
    wterm = wterm->getNext();
}

```



```

}

abag = NULL;
match = 0;
whash = utilities->hashword((const uint32_t *)stringOrder, wlength, 0xfacedeaf);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
            "**addAbag - looking for a match (hash = %i)\n", whash);
if (!(htActivityBags->empty()))
    while ((htiter = htActivityBags->find(whash)) != htActivityBags->end()) {
        abag = (htiter->second);
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "\tChecking %llx\n", (uint64_t)abag);
        // confirm that the abag you found has matching terms , otherwise
        // bump the hash by one and keep looking
        if (abag->length == wlength) {
            wterm = abag->activities;
            for (i = 0; i < abag->length; i++) stringTest[i] = 0;
            for (i = 0; i < abag->length; i++) {
                if (wterm == NULL) break;
                stringTest[i] = wterm->getTerm();
                if (parameters->debugParms >= 3)
                    fprintf(parameters->logFile,
                            "\t\tTerm = %i\n", stringTest[i]);
                wterm = wterm->getNext();
            }
            if (utilities->levenshteinI(stringOrder, wlength,
                                     stringTest, wlength) == 0) {
                match = 1;
                break;
            }
        }
        whash++;
    }
free(stringTest);
free(stringOrder);

// If the activity bag already exists
if (match) {
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "found a match\n");

    // Update the number of times the activity bag occurred

```

```

(abag->count)++;

// If attaching the timelines to the aBag, attach it
if (add_tbags) abag->timelines->qpush(wtimelines);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\t\tmatch -- timelines = %llx      %llx\n",
            (uint64_t)(abag->timelines), (uint64_t)wtimelines);

// Update the abag position counts
if (add_tables) {
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile,
            "addActivityBag -- updating abag counts\n");
    for (i = 0; i < wlength; i++) {
        if ((orderList[i] < 0) || (orderList[i] > wlength)) {
            printf("**addActivityBag -- unable to update abagCounts --- OrderLis
                i, orderList[i], wlength);
        }
        else {
            windex = i * wlength + orderList[i];
            if (windex < wlength * wlength) // should always be ok
                abag->countTable[windex]++;
        }
    }
}

// Delete the temporary variables (newTerms matches the terms in the abag)
delete qItems;
delete newTerms;
for (liter = genXref->getFront(); liter != NULL;
    liter = genXref->revIterate(liter)) {
    tmpGenXref = genXref->lvalue(liter);
    if (tmpGenXref != NULL) delete tmpGenXref;
}
delete genXref;

// Return
return whash;
}

// Otherwise, this is a new string (No aBags match)
if (parameters->debugParms >= 4)
    fprintf(parameters->logFile, "adding a new abag\n");

```

```

// Add a new activity bag
abag = new ActivityBag();

// Update status variable
(parameters->abagCount)++;

// Initialize variables
abag->isMain = 1;
abag->genericCount = 0;           // number of different generics in the aBag
if (!genXref->lempty())
    for (liter = genXref->getFront(); liter != NULL; liter = genXref->revIterate(liter))
        (abag->genericCount)++;
abag->activities = newTerms;     // the terms that make up the string
abag->length = wlength;          // number of terms
abag->count = 1;                  // how many instances of this abag exist
abag->stringID = 0;               // the string id for this aBag
if (add_tbags) {                 // the first timeline instantiation of this aBag (for
    abag->timelines->qpush(wtimelines);
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "\t\t no match -- timelines = %llx      %llx\n"
            (uint64_t)abag->timelines, (uint64_t)wtimelines);
}

// Calculate the total length of this abag (e.g., strings of strings -- use
// the length of the substrings)
wterm = newTerms;
abag->totalLength = 0;
for (i = 0; i < wlength; i++) {
    if (wterm == NULL) break;
    stringLength = strings->getTotalLength(wterm->getTerm());
    abag->totalLength += (stringLength > 0 ? stringLength : 1); // if not found, it
    wterm = wterm->getNext();
}

// If adding tables,
// add tables that do the same thing as newTerms (activity and source)
// and add a table that shows how often each activity occurred at that
// specific position in the sequence (for SMTC)
if (add_tables) {
    abag->activityTable = (int *)malloc(sizeof(int) * wlength);
    abag->sourceTable = (uint32_t *)malloc(sizeof(uint32_t) * wlength);
    abag->countTable = (uint32_t *)malloc(sizeof(uint32_t) * wlength * wlength);
}

```

```

if ((abag->activityTable == NULL) || (abag->countTable == NULL) ||
    (abag->sourceTable == NULL) || (wlength == 0)) {
    printf("**addActivityBags -- unable to allocated activity or count table --
    return 0;
}
for (i = 0; i < wlength * wlength; i++) abag->countTable[i] = 0;

// Add tables for the terms and sources
wterm = newTerms;
for (i = 0; i < wlength; i++) {
    abag->activityTable[i] = 0;
    abag->sourceTable[i] = 0;
}
for (i = 0; i < wlength; i++) {
    if (wterm == NULL) break;
    if (parameters->debugParms >= 4)
        fprintf(parameters->logFile, "\tnew term = %i\n", wterm->getTerm());
    abag->activityTable[i] = wterm->getTerm();
    abag->sourceTable[i] = wterm->getSource();
    wterm = wterm->getNext();
}

// Update the abag position counts
if (parameters->debugParms >= 4)
    fprintf(parameters->logFile, "addActivityBag -- creating count tables\n");
for (i = 0; i < wlength; i++) {
    if ((orderList[i] < 0) || (orderList[i] > wlength)) {
        printf("**addActivityBag -- unable to update abagCounts    --- OrderList[%
        i, orderList[i]);
    }
    else {
        windex = i * wlength + orderList[i];
        if (windex < wlength * wlength)
            abag->countTable[windex]++;
    }
}
} else {
    abag->activityTable = NULL;
    abag->sourceTable = NULL;
    abag->countTable = NULL;
}

// Delete temporary variables

```

```

delete qItems;
for (liter = genXref->getFront(); liter != NULL;
     liter = genXref->revIterate(liter)) {
    tmpGenXref = genXref->lvalue(liter);
    if (tmpGenXref != NULL) delete tmpGenXref;
}
delete genXref;

// Update the hash table with all the activity bags
(*htActivityBags)[whash] = abag;

// Return
return whash;
}

void ActivityBag::checkVariations(std::unordered_map<uint32_t,
                                ActivityBag *> *htActivityBags,
                                float threshold, Utilities *utilities) {
    htActivityBag_i htiter;
    ActivityBag *abag;
    int *stringOrder, *stringTest;
    int i;
    TermList *wterm;
    bool skipit;

    stringOrder = (int *)malloc(sizeof(int) * length);
    stringTest = (int *)malloc(sizeof(int) * length);
    wterm = activities;
    for (i = 0; i < length; i++) {
        if (wterm == NULL) return; // this should never happen
        stringOrder[i] = wterm->getTerm();
        wterm = wterm->getNext();
    }

    for (htiter = htActivityBags->begin(); htiter != htActivityBags->end(); htiter++)
        abag = htiter->second;
        if (abag == this) continue;
        wterm = abag->getActivities();
        skipit = false;
        for (i = 0; i < length; i++) {
            if (wterm == NULL) { skipit = true; } // this should never happen
            stringTest[i] = wterm->getTerm();
            wterm = wterm->getNext();
        }

```

```

    }
    if (skipit) continue;
    if (((float) utilities->levenshteinI(stringOrder, length, stringTest,
        length) / (float) length) <= threshold) {
        isMain = 0;
        mainActivityBag = abag;
        return;
    }
}

}

uint8_t ActivityBag::createString(std::unordered_map<uint32_t,
    ActivityBag *> *htActivityBags, Strings *strings) {
    htActivityBag_i htiter;
    ActivityBag *abag;
    uint32_t wprimaryid;

    for (htiter = htActivityBags->begin(); htiter != htActivityBags->end(); htiter++)
        abag = htiter->second;
    if (abag->getRemove() == 1 || abag->getLength() <= 1) continue;
    wprimaryid = 0;
    if (!(abag->getIsMain())) {
        if (abag->getMainActivityBag() != NULL) {
            wprimaryid = abag->getMainActivityBag()->getStringID();
        }
        if (wprimaryid != 0) {
            abag->setStringID(strings->createString(abag->getActivities(),
                abag->getLength(), 0));
        }
    }
    else {
        abag->setStringID(strings->createString(abag->getActivities(),
            abag->getLength(), 0));
    }
}

for (htiter = htActivityBags->begin(); htiter != htActivityBags->end(); htiter++)
    abag = htiter->second;
if (abag->getRemove() == 1 || abag->getLength() <= 1) continue;
if (abag->getStringID() != 0) continue;
if (abag->getIsMain()) continue;
if (abag->getMainActivityBag() == NULL) continue;

```

```

        wprimaryid = abag->getMainActivityBag()->getStringID();
        if (wprimaryid != 0) { // this should always be the case
            abag->setStringID(strings->createString(abag->getActivities(),
                                                    abag->getLength(), 0));
        }
    }
    return 1;
}

```

Appendix G. Events

7.1 Timeline

Header.

```
#pragma once
#include "TearCPP.h"
#include "ItemInstance.h"
#include "Terms.h"
#include "Patterns.h"
#include "Generics.h"
#include "TimelineEntry.h"
#include "Utilities.h"
class TimelineEntry;

class Timeline
{
    Utilities *utilities;
    Parameters *parameters;
    Generics *generics;
    Terms *terms;
    sqlite3_stmt *stmt_timeline;
    sqlite3_stmt *stmt_selActivity;
    sqlite3_stmt *stmt_selStringInstance;
    sqlite3_stmt *stmt_selStringSupport;
    sqlite3_stmt *stmt_selStringSupportSingle;
    sqlite3_stmt *stmt_selStringSupportXref;
    sqlite3_stmt *stmt_selLoop;
    sqlite3_stmt *stmt_selLoopSupport;

    // One of these two is used to iterate
    char *caseList;
    void *dateRangeIter;

    // Variables to track and report elapsed time
    struct tm timeinfo;
    time_t wtimer;
    char showtime[80];
    uint32_t last_seen, ctr;

public:
    Timeline(Utilities *wutilities, Parameters *wparameters, Terms *wterms,
            Generics *wgenerics);
```



```

~Timeline();
char *initialize();
TimelineEntry *process(uint8_t skipLoops);
char *iterate();
ItemInstance *getStringItems(uint32_t wactivity, uint32_t *wactor);
ItemInstance *getLoopItems(uint32_t wloopid, uint32_t *wactor);
int getActivity(uint32_t wactivity, uint32_t *wactor, uint32_t *waction,
                uint32_t *wappexec, uint32_t *witemtype, ItemInstance **witems,
                int *wtermT);
};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Timeline                                                                                   //
// Input: date and case values from parameter table                                             //
// Output: the events that match                                                                 //
// Notes:                                                                                           //
// Initial Release: 29-Sep-17                                                                    //
//                                                                                               //
// Revision History:                                                                               //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "Timeline.h"
Timeline::~Timeline() {}

```

```

Timeline::Timeline(Utilities *wutilities, Parameters *wparameters,
                  Terms *wterms, Generics *wgenerics)
{
    char exec_string[2000];
    int rc;

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Initialize Variables
utilities = wutilities;
parameters = wparameters;
terms = wterms;
generics = wgenerics;
dateRangeIter = NULL;

// There are two possibilities --

```

```

//      1. The parameter caseList isn't blank -- in that case use it
//      2. The parameters caseList is blank -- use the parameter dateRanges
if (parameters->scaseList != NULL) {
    caseList = (char *)malloc(sizeof(char) * (strlen(parameters->scaseList) + 2));
    strcpy_s(caseList, strlen(parameters->scaseList) + 1, parameters->scaseList);
}

// Create the SQL Statements
if (parameters->existingLevel == 10) // every record has a link to an item
    sprintf_s(exec_string, 2000, "SELECT timelineID, timelinetypeid, activitylid, t
else
    sprintf_s(exec_string, 2000, "SELECT timelineID, timelinetypeid, activitylid, t
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_timeline, 0);

sprintf_s(exec_string, 2000, "SELECT actorid, actionid, applicationid, items.itemt
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selActivity, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Timeline Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT stringID from stringInstances where instanceI
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selStringInstance
if (rc != SQLITE_OK) {
    printf("Error: %i: couldn't prepare Select String Instance Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT genericType, genericValue from stringSupport
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selStringSupport,
if (rc != SQLITE_OK) {
    printf("Error: %i: couldn't prepare Select String Support Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT newGenericid FROM StringSupportXref WHERE ins
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selStringSupportX
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select StringSupport Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT genericValue from StringSupport WHERE instanc

```

```

rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selStringSupportS
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select StringSupport Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT termID, sourceID FROM loops WHERE loopid = ?1
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selLoop, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i: couldn't prepare Select Loop Statement\n", rc);
    return;
}

sprintf_s(exec_string, 2000, "SELECT activity1id, sourceID FROM loopSupport, timel
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selLoopSupport, 0
if (rc != SQLITE_OK) {
    printf("Error: %i: couldn't prepare Select Loop Support Statement\n", rc);
    return;
}

}

char *Timeline::initialize() {
    // Purpose: get the first set of values of the events and start the SQL Statement

    // Variables for handling the list of cases
    char *wcomma, *scaseNbr;
    Parameters::dateRange_t *wdateRange;

    // Progress variables
    time_t wtimer;
    struct tm timeinfo;
    char stime[80], etime[80];

    //////////////////////////////////////

    // Initialize variables
    ctr = 0;
    last_seen = 0;

    // If the date range isn't empty, get the first date range (and case number)
    // from it and stuff them into parameters->startDate, endDate, and CaseList --

```

```

// makes the rest of the code simple
if (!(parameters->dateRange->empty())) {
    dateRangeIter = parameters->dateRange->getFront();

    wdateRange = parameters->dateRange->lvalue(dateRangeIter);
    parameters->startDate = wdateRange->startDate;
    parameters->endDate = wdateRange->endDate;
    parameters->scaseList = wdateRange->caseNbr;

    // Reset the SQL statement and update the values
    sqlite3_reset(stmt_timeline);
    sqlite3_bind_text(stmt_timeline, 1, wdateRange->caseNbr, -1, SQLITE_TRANSIENT);
    sqlite3_bind_int64(stmt_timeline, 2, (uint64_t) wdateRange->startDate);
    sqlite3_bind_int64(stmt_timeline, 3, (uint64_t) wdateRange->endDate);

    wtimer = (time_t) wdateRange->startDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(stime, 80, "%x %X", &timeinfo);

    wtimer = (time_t) wdateRange->endDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(etime, 80, "%x %X", &timeinfo);

    // Show the user the current parameters
    printf("Case %s (%s - %s)\n", wdateRange->caseNbr, stime, etime);

    // Return the case number
    return wdateRange->caseNbr;
}

// Peel the next case number off the case list
printf("\n\nCase List = %s\n\n", caseList);
wcomma = (caseList == NULL ? NULL : strchr(caseList, ','));
if (caseList == NULL)
    scaseNbr = NULL;
else {
    scaseNbr = (char *)malloc(sizeof(char) * (
        (wcomma == NULL ? strlen(caseList) : wcomma - caseList) + 2));
    strncpy_s(scaseNbr,
        (wcomma == NULL ? strlen(caseList) : wcomma - caseList) + 1,
        caseList, (wcomma == NULL ? strlen(caseList) : wcomma - caseList));
    caseList = (wcomma == NULL ? NULL : wcomma + 1);
}

```

```

// If there are no more cases, we're done
if (scaseNbr == NULL) return NULL;

// Reset the SQL statement and update the values
sqlite3_reset(stmt_timeline);
sqlite3_bind_text(stmt_timeline, 1, scaseNbr, -1, SQLITE_TRANSIENT);
sqlite3_bind_int64(stmt_timeline, 2, (uint64_t) parameters->startDate);
sqlite3_bind_int64(stmt_timeline, 3, (uint64_t) parameters->endDate);

// Show the user the current parameters
wtimer = (time_t)parameters->startDate;
localtime_s(&timeinfo, &wtimer);
strftime(stime, 80, "%x %X", &timeinfo);

wtimer = (time_t)parameters->endDate;
localtime_s(&timeinfo, &wtimer);
strftime(etime, 80, "%x %X", &timeinfo);

printf("Case %s (%s - %s)\n", scaseNbr, stime, etime);
return scaseNbr;
}

char *Timeline::iterate() {
    // Purpose: get the next set of values of the events and start the SQL Statement

    // Variables for handling the list of cases
    char *scaseNbr, *wcomma;
    Parameters::dateRange_t *wdateRange;

    // Progress variables
    time_t wtimer;
    struct tm timeinfo;
    char stime[80], etime[80];

    //////////////////////////////////////

    // Initialize the variables
    ctr = 0;
    last_seen = 0;

```

```

if (dateRangeIter != NULL) {
    dateRangeIter = parameters->dateRange->revIterate(dateRangeIter);
    if (dateRangeIter == NULL) return NULL;

    wdateRange = parameters->dateRange->lvalue(dateRangeIter);
    parameters->startDate = wdateRange->startDate;
    parameters->endDate = wdateRange->endDate;
    parameters->scaseList = wdateRange->caseNbr;

    // Reset the SQL statement and update the values
    sqlite3_reset(stmt_timeline);
    sqlite3_bind_text(stmt_timeline, 1, wdateRange->caseNbr, -1, SQLITE_TRANSIENT);
    sqlite3_bind_int64(stmt_timeline, 2, (uint64_t) parameters->startDate);
    sqlite3_bind_int64(stmt_timeline, 3, (uint64_t) parameters->endDate);

    // Show the user the current parameters
    wtimer = (time_t)wdateRange->startDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(stime, 80, "%x %X", &timeinfo);

    wtimer = (time_t)wdateRange->endDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(etime, 80, "%x %X", &timeinfo);

    printf("Case %s (%s - %s)\n", wdateRange->caseNbr, stime, etime);

    // Return the case number
    return wdateRange->caseNbr;
}

// Peel the next case number off the case list
scaseNbr = NULL;
wcomma = (caseList == NULL ? NULL : strchr(caseList, ','));
if (caseList == NULL)
    scaseNbr = NULL;
else {
    scaseNbr = (char *)malloc(sizeof(char) * (
        (wcomma == NULL ? strlen(caseList) : wcomma - caseList) + 2));
    if (scaseNbr != NULL) {
        strncpy_s(scaseNbr, (wcomma == NULL ? strlen(caseList) : wcomma - caseList),
            caseList, (wcomma == NULL ? strlen(caseList) : wcomma - caseList));
        caseList = (wcomma == NULL ? NULL : wcomma + 1);
    }
}

```

```

}

// If there are no more cases, we're done
if (scaseNbr == NULL) return NULL;

// Reset the SQL statement and update the values
sqlite3_reset(stmt_timeline);
sqlite3_bind_text(stmt_timeline, 1, scaseNbr, -1, SQLITE_TRANSIENT);
sqlite3_bind_int64(stmt_timeline, 2, (uint64_t) parameters->startDate);
sqlite3_bind_int64(stmt_timeline, 3, (uint64_t) parameters->endDate);

// Show the user the current parameters
wtimer = (time_t)parameters->startDate;
localtime_s(&timeinfo, &wtimer);
strftime(stime, 80, "%x %X", &timeinfo);

wtimer = (time_t)parameters->endDate;
localtime_s(&timeinfo, &wtimer);
strftime(etime, 80, "%x %X", &timeinfo);

printf("Case %s (%s - %s)\n", scaseNbr, stime, etime);

// Return the case number
return scaseNbr;
}

TimelineEntry *Timeline::process(uint8_t skipLoops) {
    // Purpose: return the next event record

    TimelineEntry *entry;
    int rc;
    uint32_t wsourceid;
    uint32_t wactor, waction, wappexec, witemtype;
    int wtermT;
    ItemInstance *witems;
    TermList *wterm;

    //////////////////////////////////////

    // Grab the next event record from the DB
    rc = sqlite3_step(stmt_timeline);
    if (rc != SQLITE_ROW) return NULL;

```

```

// Show the progress
if (utilities->showTime(0, &ctr, &last_seen, parameters->increment))
    printf("\t\tTermcount = %i\t\tActivity Bag Count = %u\n",
           (int)parameters->termCount, parameters->abagCount);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "rc = %i    timelineid = %i    activityid = %i\n"
        rc, sqlite3_column_int(stmt_timeline, 0), sqlite3_column_int(stmt_timeline,

// If we're skipping loops, keep grabbing records till you find one that isn't a l
// TODO: make sure SMTc and DVSMT can handle loops
if (skipLoops) {
    while (sqlite3_column_int(stmt_timeline, 5) == S_LOOP) {
        rc = sqlite3_step(stmt_timeline);
        if (rc != SQLITE_ROW) return NULL;
    }
}

// Create a new instance to store the current event
entry = new TimelineEntry(sqlite3_column_int(stmt_timeline, 1), //timelinetype
    sqlite3_column_int(stmt_timeline, 2), //activity
    sqlite3_column_int(stmt_timeline, 7), // relationships
    sqlite3_column_int(stmt_timeline, 3), // startTime
    sqlite3_column_int(stmt_timeline, 4), // endTime
    sqlite3_column_int(stmt_timeline, 5), // sourceID
    0, // traceType
    (char *)sqlite3_column_text(stmt_timeline, 8), // caseNbr
    sqlite3_column_int(stmt_timeline, 6), // traceid
    sqlite3_column_int(stmt_timeline, 0) // timelineid);
    );

// If the end time is weird (e.g., missing), set it to the start time
if (entry->getEndTime() < entry->getStartTime())
    entry->setEndTime(entry->getStartTime());

// Get the items for the event, this will vary based on the time of event
if (entry->getSourceID() == S_STRING) {

    // Grab the term for this string instance
    sqlite3_reset(stmt_selStringInstance);
    sqlite3_bind_int(stmt_selStringInstance, 1, entry->getActivity());
    rc = sqlite3_step(stmt_selStringInstance);
    if (rc != SQLITE_ROW) {

```



```

        printf("**process -- couldn't find string instance %u -- error %i -- returni
        delete entry;
        return NULL;
    }

    // Set the term
    entry->setTerm(sqlite3_column_int(stmt_selStringInstance, 0));

    // Get the items
    entry->setItems(getStringItems(entry->getActivity(), &wactor));

    // Set the actor
    entry->setActor(wactor);

    // If there are no items, don't return anything
    if (entry->getItems() == NULL) {
        printf("**process -- couldn't find string support %u -- returning\n",
                entry->getActivity());
        delete entry;
        return NULL;
    }
}
else if (entry->getSourceID() == S_LOOP) {

    // Grab the term for this loop
    sqlite3_reset(stmt_selLoop);
    sqlite3_bind_int(stmt_selLoop, 1, entry->getActivity());
    rc = sqlite3_step(stmt_selLoop);
    if (rc != SQLITE_ROW) {
        printf("**process** couldn't find loop %u -- error %i -- returning\n",
                entry->getActivity(), rc);
        delete entry;
        return NULL;
    }

    // Set the term and source (e.g., loop of terms, loop of strings)
    wtermT = sqlite3_column_int(stmt_selLoop, 0);
    wsourceid = sqlite3_column_int(stmt_selLoop, 1);
    if ((wsourceid != S_STRING)) {
        entry->setTermInfo(wtermT, terms->getActor(wtermT),
                terms->getAction(wtermT), terms->getAppexec(wtermT),
                terms->getItemtype(wtermT),
                getLoopItems(entry->getActivity(), &wactor));
    }
}

```

```

    } else {
        entry->setTerm(wtermT);
        entry->setItems(getLoopItems(entry->getActivity(), &wactor));
        entry->setActor(wactor);
    }
}
else {
    // Get the details from the activity
    witems = NULL;
    getActivity(entry->getActivity(), &wactor, &waction, &wappexec,
                &witemtype, &witems, &wtermT);
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile, "**timeline:process -- term = %i\n", wtermT);
    entry->setTermInfo(wtermT, wactor, waction, wappexec, witemtype, witems);
}

// Create a termList for faster processing in addActivityBag later
wterm = new TermList(entry->getTerm(), entry->getSourceID(), 0, NULL, NULL);
entry->setTermList(wterm);

// Return the event record
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
            "term = %i    source = %i\n", wterm->getTerm(), wterm->getSource());
return entry;
}

ItemInstance *Timeline::getLoopItems(uint32_t wloopid, uint32_t *wactor) {
    // Purpose: get the loop items for this loopInstance

    int rc;
    uint32_t wactivityid, wsourceid, localActor;
    ItemInstance *itemsContainer, *retItem;
    listT<ItemInstance> *wloopList;

    //////////////////////////////////////

    // Actor is the same for the entire loop, so whichever actor we get is fine
    *wactor = 0;

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "**getLoopItems - loopid = %i\n", wloopid);
    sqlite3_reset(stmt_selLoopSupport);

```

```

sqlite3_bind_int(stmt_selLoopSupport, 1, wloopid);
wloopList = new listT<ItemInstance>;

// Iterate through all the events for this loop
while ((rc = sqlite3_step(stmt_selLoopSupport)) == SQLITE_ROW) {
    // Get the associated activity id and source
    wactivityid = sqlite3_column_int(stmt_selLoopSupport, 0);
    wsourceid = sqlite3_column_int(stmt_selLoopSupport, 1); // should be same

    // If the source is strings, get the string items
    if (wsourceid == S_STRING) {
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, " getLoopItems -- adding a string item\n");
        itemsContainer = getStringItems(wactivityid, &localActor);
        if (localActor != 0) *wactor = localActor;
    }
    else if (wsourceid == S_LOOP) { // shouldn't happen -- no loops of loops
        printf("**getLoopItems -- loop of loops\n");
        itemsContainer = getLoopItems(wactivityid, &localActor);
        if (localActor != 0) *wactor = localActor;
    }
    else {
        // Otherwise, get the items associated with this activity
        itemsContainer = new ItemInstance();
        if (itemsContainer == NULL) {
            printf("**getLoopItems -- unable to allocate itemsContainer -- returning\n");
            return NULL;
        }
        if (parameters->debugParms >= 6)
            fprintf(parameters->logFile, " getLoopItems -- adding an activity\n");
        sqlite3_reset(stmt_selActivity);
        sqlite3_bind_int(stmt_selActivity, 1, wactivityid);
        rc = sqlite3_step(stmt_selActivity);
        if (rc == SQLITE_ROW) {
            localActor = (uint32_t)sqlite3_column_int(stmt_selActivity, 0);
            if (localActor != 0) *wactor = localActor;
            itemsContainer->setGenericType(GNRC_ITEM);
            itemsContainer->setItem(
                (uint32_t)sqlite3_column_int(stmt_selActivity, 4));
        }
        else {
            printf("**getLoopItems -- unable to find activity %u -- returning\n", wactivityid);
            return NULL;
        }
    }
}

```

```

    }
    if (parameters->debugParms >= 6)
        fprintf(parameters->logFile, " getLoopItems -- adding an activity\n");
}

// Add the current event's item(s) to the end of the loopList
wloopList->insertRear(itemsContainer);
}

// Now, create a blank ItemInstance that shows it's a loop
// --- genericType = GNRC_ITEMLIST, listID = loopID (i.e., not zero),
// and loopList is populated
retItem = new ItemInstance();
retItem->setGenericType(GNRC_ITEMLIST);
retItem->setListID(wloopid);
retItem->setLoopList(wloopList);

return retItem;
}

int Timeline::getActivity(uint32_t wactivity, uint32_t *wactor, uint32_t *waction,
    uint32_t *wappexec, uint32_t *witemtype, ItemInstance **witems,
    int *wtermT) {
    // Purpose: get the information for the passed activity

    int rc;

    //////////////////////////////////////

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "**getActivity - activity = %i\n", wactivity);
    sqlite3_reset(stmt_selActivity);
    sqlite3_bind_int(stmt_selActivity, 1, wactivity);
    rc = sqlite3_step(stmt_selActivity);
    if (rc == SQLITE_ROW) {
        *witems = new ItemInstance((uint32_t)sqlite3_column_int(stmt_selActivity, 4), /
            GNRC_ITEM);
        *wactor = (uint32_t)sqlite3_column_int(stmt_selActivity, 0);
        *waction = (uint32_t)sqlite3_column_int(stmt_selActivity, 1);
        *wappexec = (uint32_t)sqlite3_column_int(stmt_selActivity, 2);
        *witemtype = (uint32_t)sqlite3_column_int(stmt_selActivity, 3);
        *wtermT = terms->getGenericTermA(*wactor, *waction, *wappexec, *witemtype,
            (*witems)->getItem()); // -1 if error
    }
}

```

```

        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile,
                "\tactor = %i    action = %i    app = %i    type = %i    item = %i    term = %i\n",
                *wactor, *waction, *wappexec, *witemtype, (*witems)->getItem(), *wtermT);
        return 1;
    }
    else {
        *wactor = *waction = *wappexec = *witemtype = *wtermT = 0;
        *witems = NULL;
        if (parameters->debugParms >= 5) fprintf(parameters->logFile, "\tretaining null\n");
        return 0;
    }
    return 1;
}

```

```

ItemInstance *Timeline::getStringItems(uint32_t wactivity, uint32_t *wactor) {
    // Purpose: Get the items for this string

    int rc;
    ItemInstance *witems, *prevItemInstance, *firstItemInstance, *tmpItem;
    uint32_t wgenericType;
    uint32_t localActor;

    //////////////////////////////////////

    *wactor = 0;

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "**getStringItems -- activity = %u\n", wactivity);
    sqlite3_reset(stmt_selStringSupport);
    sqlite3_bind_int(stmt_selStringSupport, 1, wactivity);
    rc = sqlite3_step(stmt_selStringSupport);
    if (rc != SQLITE_ROW) {
        printf("**getStringItems -- couldn't find string support %u -- error %i -- returning NULL\n", wactivity, rc);
        return NULL;
    }
    prevItemInstance = NULL;
    firstItemInstance = NULL;

    // Iterate through all the string support (i.e., the generic item
    // assignments) for this event
    while (rc == SQLITE_ROW) {
        wgenericType = sqlite3_column_int(stmt_selStringSupport, 0);

```

```

// Note: it's possible that GNRC_ACTOR is populated if the actor isn't
// the system
if (wgenericType == GNRC_ACTOR) {
    *wactor = sqlite3_column_int(stmt_selStringSupport, 1);
    rc = sqlite3_step(stmt_selStringSupport);
    continue;
}

// Create a separate item instance for each different generic item
witems = new ItemInstance();

witems->setGenericType(wgenericType);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\tGeneric Type = %i\n",
            witems->getGenericType());

// If it's a generic item, just set it in the itemInstance
if (wgenericType == GNRC_ITEM) {
    witems->setItem(sqlite3_column_int(stmt_selStringSupport, 1));
    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "\tItem = %i\n",
                witems->getItem());
}
else if (wgenericType == GNRC_ITEMLIST) {
    // otherwise, it's a loop and set the listID and the then get items
    // why can't it be a string of strings?
    // -- it can, but we store the assignments of generic items
    // for this instance in a single level at the top
    witems->setListID(atoi(
        (const char *)sqlite3_column_text(stmt_selStringSupport, 1)));
    witems->setLoopList(
        getLoopItems(witems->getListID(), &localActor)->getLoopList());
}
if (prevItemInstance != NULL) prevItemInstance->setNext(witems);
prevItemInstance = witems;

if (firstItemInstance == NULL) firstItemInstance = witems;
rc = sqlite3_step(stmt_selStringSupport);
if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\t%u %llx\n",
            witems->getItem(), (uint64_t)witems->getItemList());

```

```

}

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
        "**getStringItems -- firstItem = %llx\n", (uint64_t)firstItemInstance);

// Now, create a blank ItemInstance that shows it's a string
//      --- genericType = GNRC_ITEMLIST, listID = zero, and itemList is populated
tmpItem = new ItemInstance();
tmpItem->setGenericType(GNRC_ITEMLIST);
tmpItem->setListID(0);
tmpItem->setItemList(firstItemInstance);

return tmpItem;
}

```

7.2 TimelineEntry

Header.

```

#pragma once
#include "TearCPP.h"
#include "Timeline.h"

class TimelineEntry
{
    uint32_t timelinetypeid;
    ItemInstance *items; // used to figure genericItems needed in calcString
    TermList *termL; // just speeds up processing in addActivityBag and AbstractString
    uint32_t relationship;
    uint32_t activity;
    uint32_t traceid;
    uint32_t tracetype;
    uint32_t order; // used in smtc PowerHelper
    uint32_t startTime;
    uint32_t endTime;
    uint32_t sourceid;

    uint32_t actor;
    uint32_t action;
    uint32_t appexec;
    uint32_t itemtype;
}

```

```

char *caseNbr;

TimelineEntry *prev;
TimelineEntry *next;

public:
    uint32_t timelineid; // public for smtc comparison to run faster
    int term;           // public for smtc comparison to run faster

    ~TimelineEntry() {
        if (caseNbr != NULL) free(caseNbr);
        if (items != NULL) delete items;
        if (termL != NULL) termL->erase();
    }

    TimelineEntry() :
        timelinetypeid(0), timelineid(0), items(NULL), term(-1), relationship(0),
        activity(0), traceid(0), tracetype(0), order(0), startTime(0), endTime(0),
        sourceid(0), actor(0), action(0), appexec(0), itemtype(0),
        caseNbr(NULL), prev(NULL), next(NULL), termL(NULL) {}

    TimelineEntry(uint32_t wtimelinetype, uint32_t wactivity, uint32_t wrelationship,
        uint32_t wstartTime, uint32_t wendTime, uint32_t wsourceid, uint32_t wtracetype,
        char *wcaseNbr, uint32_t wtraceid) :
        timelinetypeid(wtimelinetype), timelineid(0), items(NULL), term(-1),
        relationship(wrelationship), activity(wactivity),
        traceid(wtraceid), tracetype(wtracetype), order(0),
        startTime(wstartTime), endTime(wendTime), sourceid(wsourceid),
        actor(0), action(0), appexec(0), itemtype(0),
        caseNbr(NULL), prev(NULL), next(NULL), termL(NULL)
    {
        if (wcaseNbr != NULL) {
            caseNbr = (char *)malloc(sizeof(char) * (strlen(wcaseNbr) + 2));
            strcpy_s(caseNbr, strlen(wcaseNbr) + 1, wcaseNbr);
        }
    }

    TimelineEntry(uint32_t wtimelinetype, uint32_t wactivity, uint32_t wrelationship,
        uint32_t wstartTime, uint32_t wendTime, uint32_t wsourceid, uint32_t wtracetype,
        char *wcaseNbr, uint32_t wtraceid, uint32_t wtimelineid) :
        timelinetypeid(wtimelinetype), timelineid(wtimelineid), items(NULL), term(-1),
        relationship(wrelationship), activity(wactivity),
        traceid(wtraceid), tracetype(wtracetype), order(0),

```



```

    startTime(wstartTime), endTime(wendTime), sourceid(wsourceid),
    actor(0), action(0), appexec(0), itemtype(0),
    caseNbr(NULL), prev(NULL), next(NULL), termL(NULL)
{
    if (wcaseNbr != NULL) {
        caseNbr = (char *)malloc(sizeof(char) * (strlen(wcaseNbr) + 2));
        strcpy_s(caseNbr, strlen(wcaseNbr) + 1, wcaseNbr);
    }
}

uint8_t setSourceID(uint32_t wsourceid) { sourceid = wsourceid; return 1; }
uint8_t setTimeline(uint32_t wtimeline) { timelineid = wtimeline; return 1; }
uint8_t setStartTime(uint32_t wstime) { startTime = wstime; return 1; }
uint8_t setEndTime(uint32_t wetime) { endTime = wetime; return 1; }
uint8_t setRelationship(uint32_t wrelationship) {
    relationship = wrelationship;
    return 1;
}
uint8_t setActivity(uint32_t wactivity) { activity = wactivity; return 1; }
uint8_t setTerm(int wterm) { term = wterm; return 1; }
uint8_t setActor(uint32_t wactor) { actor = wactor; return 1; }
uint8_t setAction(uint32_t waction) { action = waction; return 1; }
uint8_t setAppexec(uint32_t wappexec) { appexec = wappexec; return 1; }
uint8_t setItemtype(uint32_t witemtype) { itemtype = witemtype; return 1; }
uint8_t setItems(ItemInstance *witems) { items = witems; return 1; }
uint8_t TimelineEntry::setTermInfo(int wtermT, uint32_t wactor, uint32_t waction,
    uint32_t wappexec, uint32_t witemtype, ItemInstance *witems) {
    term = wtermT; actor = wactor; action = waction; appexec = wappexec;
    itemtype = witemtype; items = witems; return 1;
}
uint8_t setPrev(TimelineEntry *wprev) { prev = wprev; return 1; }
uint8_t setNext(TimelineEntry *wnext) { next = wnext; return 1; }
uint8_t setOrder(uint32_t worder) { order = worder; return 1; }
uint8_t setTermList(TermList *wtermL) { termL = wtermL; return 1; }
uint32_t getSourceID() { return sourceid; }
const uint32_t getTimelineID() { return timelineid; }
uint32_t getStartTime() { return startTime; }
uint32_t getEndTime() { return endTime; }
uint32_t getRelationship() { return relationship; }
uint32_t getActivity() { return activity; }
int getTerm() { return term; }
uint32_t getActor() { return actor; }
uint32_t getAction() { return action; }

```

```

uint32_t getAppexec() { return appexec; }
uint32_t getItemtype() { return itemtype; }
uint32_t getOrder() { return order; }
char *getCaseNbr() { return caseNbr; }
uint32_t getTraceType() { return tracetype; }
uint32_t getTraceid() { return traceid; }
ItemInstance *getItems() { return items; }
TimelineEntry *getPrev() { return prev; }
TimelineEntry *getNext() { return next; }
TermList *getTermList() { return termL; }

uint8_t setCasenbr(char *wcaseNbr) {
    if (wcaseNbr == NULL) {
        caseNbr = NULL;
        return 0;
    }
    caseNbr = (char *)malloc(sizeof(char) * (strlen(wcaseNbr) + 2));
    strcpy_s(caseNbr, strlen(wcaseNbr) + 1, wcaseNbr);
    return 1;
}
};

```

7.3 TermList

Header.

```
#pragma once
#include "TearCPP.h"

class TermList
{
    int term;
    uint32_t sourceid;
    uint32_t order;
    char *orderid;
    TermList *next;

public:
    TermList() : term(0), sourceid(0), order(0), orderid(NULL), next(NULL) {}
    TermList(int wterm, uint32_t wsourceid, uint32_t worder, char *worderid,
              TermList *wnext) :
        term(wterm), sourceid(wsourceid), order(worder), orderid(NULL),
        next(wnext) {

        if (worderid != NULL) {
            orderid = (char *)malloc(sizeof(char) * (strlen(worderid) + 2));
            strcpy_s(orderid, strlen(worderid) + 1, worderid);
        }
    }

    ~TermList() {
        if ((uint64_t)this <= 0x7fffffff && (uint64_t)this > 0) {
            if (orderid != NULL) free(orderid);
            if (next != NULL && (uint64_t)next <= 0x000000007fffffff &&
                (uint64_t)next > 0) {
                delete next;
            }
        }
        else return;
    };

    void erase() {
        next = NULL;
        delete this;
    }
}
```

```

int getTerm() { if (this != NULL) return term; else return -1; }
uint32_t getSource() { return sourceid; }
uint32_t getOrder() { return order; }
char *getOrderid() { return orderid; }
TermList *getNext() { return next; }

TermList *TermList::getTail() {
    TermList *wterms, *wtermTail;
    wtermTail = this;
    wterms = next;
    while (wterms != NULL) { wtermTail = wterms;    wterms = wterms->next; }
    return wtermTail;
}

uint8_t setTerm(int wterm) { term = wterm; return 1; }
uint8_t setSourceid(uint32_t wsourceid) { sourceid = wsourceid; return 1; }
uint8_t setOrder(uint32_t worder) { order = worder; return 1; }
uint8_t setNext(TermList *wnext) { next = wnext; return 1; }
uint8_t setOrderid(char * worderid) {
    if (orderid != NULL) free(orderid);
    if (worderid == NULL) {
        orderid = NULL; return 1;
    }
    orderid = (char *)malloc(sizeof(char) * (strlen(worderid) + 2));
    strcpy_s(orderid, strlen(worderid) + 1, worderid);
    return 1;
}

};

```

7.4 ItemInstance

Header.

```
#pragma once
#include "TearCPP.h"
#include "lists.h"
#include "ItemUtilities.h"

class ItemInstance
{
    unsigned int genericType;
    unsigned int itemid;
    unsigned int listid;
    unsigned char inDB;
    ItemInstance *itemList;
    listT<ItemInstance> *loopList;
    ItemInstance *prev;
    ItemInstance *next;
    unsigned int nextCount;
    unsigned int listCount;

    //////////////////////////////////////
    // MATCH ROUTINE -- duplicated to avoid pass parameters around

    /* matchstar: search for c*regexp at beginning of text */
    int ItemInstance::matchstar(int c, char *regexp, char *text)
    {
        do { /* a * matches zero or more instances */
            if (matchhere(regexp, text))
                return 1;
        } while (*text != '\0' && (*text++ == c || c == '.'));
        return 0;
    }

    /* matchhere: search for regexp at beginning of text */
    int ItemInstance::matchhere(char *regexp, char *text)
    {
        if (regexp[0] == '\0')
            return 1;
        if (regexp[1] == '*')
            return matchstar(regexp[0], regexp + 2, text);
        if (regexp[0] == '$' && regexp[1] == '\0')
```

```

        return *text == '\0';
    if (*text != '\0' && (regex[0] == '.' || regex[0] == *text))
        return matchhere(regex + 1, text + 1);
    return 0;
}

/* match: search for regex anywhere in text */
int ItemInstance::match(char *regex, char *text)
{
    if ((regex == NULL) || (text == NULL)) return 1;

    if (regex[0] == '^')
        return matchhere(regex + 1, text);
    do { /* must look even if string is empty */
        if (matchhere(regex, text))
            return 1;
    } while (*text++ != '\0');
    return 0;
}

// MATCH ROUTINE -- duplicated to avoid pass parameters around
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

public:
    ItemInstance::ItemInstance() :
        listCount(0), nextCount(0), genericType(0), inDB(0), itemid(0), listid(0),
        loopList(NULL), itemList(NULL), prev(NULL), next(NULL)
    {}

    ItemInstance::ItemInstance(unsigned int witemid, unsigned int wgenericType) :
        itemid(witemid), genericType(wgenericType), nextCount(0), inDB(0), listid(0),
        loopList(NULL), itemList(NULL), prev(NULL), next(NULL) {
    }

    ItemInstance::ItemInstance(ItemInstance *tmp)
    {
        if (tmp == NULL) {
            listCount = 0;
            nextCount = 0;
            genericType = 0;
            inDB = 0;
            itemid = 0;
            listid = 0;
            loopList = NULL;
            itemList = NULL;

```

```

        prev = NULL;
        next = NULL;
    }

    genericType = tmp->getGenericType();
    itemid = tmp->getItem();
    listid = tmp->getListID();
    inDB = tmp->getDB();

    if (tmp->itemList != NULL)
        itemList = (tmp->itemList)->itemInstanceCopy();
    else
        itemList = NULL;

    if (tmp->getNext() != NULL) {
        next = (tmp->getNext())->itemInstanceCopy();
        next->prev = this;
    }

    if (tmp->getLoopList() != NULL) {
        loopList = tmp->getLoopList()->copyClass();
    }
    else
        loopList = NULL;

    nextCount = tmp->getNextElementCount();
    listCount = tmp->getListElementCount();
}

ItemInstance::~ItemInstance()
{
    void *liter;
    ItemInstance *witem;

    if (itemList != NULL) delete itemList;
    if (next != NULL) delete next;
    if (loopList != NULL) {
        for (liter = loopList->getFront(); liter != NULL;
             liter = loopList->revIterate(liter)) {
            witem = loopList->lvalue(liter);
            delete witem;
        }
    }
}

```

```

        }
        delete loopList;
    }
}

// Set Values
unsigned char setGenericType(unsigned long wgenericType) {
    genericType = wgenericType;
    return 1;
}

unsigned char setItem(unsigned long witemid) { itemid = witemid; return 1; }
unsigned char setListID(unsigned long wlistid) { listid = wlistid; return 1; }
unsigned char setDB(unsigned char winDB) { inDB = winDB; return 1; }
unsigned char setItemList(ItemInstance *wlist) { itemList = wlist; return 1; }
unsigned char setLoopList(listT<ItemInstance> *wlist) { loopList = wlist; return 1; }

// Get Values
ItemInstance *getItemList() { return itemList; }
listT<ItemInstance> *getLoopList() { return loopList; }
unsigned int getGenericType() { return genericType; }
unsigned int getItem() { return itemid; }
unsigned int getListID() { return listid; }
unsigned char getDB() { return inDB; }
ItemInstance *getNext() { return next; }
unsigned int getNextElementCount() { return nextCount; }
unsigned int getListElementCount() { return listCount; }

ItemInstance *ItemInstance::itemInstanceCopy() {

    ItemInstance *witem, *wptr, *whead, *wtail;

    if (this == NULL) return NULL;
    witem = new ItemInstance();
    witem->setGenericType(genericType);
    witem->setItem(itemid);
    witem->setListID(listid);
    witem->setDB(inDB);
    witem->nextCount = nextCount;
    witem->listCount = listCount;
    if (loopList != NULL)
        witem->loopList = loopList->copyClass();
    else
        witem->loopList = NULL;
}

```



```

    if (itemList != NULL)
        witem->itemList = itemList->itemInstanceCopy();
    else
        witem->itemList = NULL;
    if (next != NULL) {
        wptr = next;
        whead = witem;
        wtail = witem;
        while (wptr != NULL) {
            witem = new ItemInstance();
            witem->setGenericType(wptr->genericType);
            witem->setItem(wptr->itemid);
            witem->setListID(wptr->listid);
            witem->setDB(wptr->inDB);
            witem->prev = wtail;
            witem->nextCount = wptr->nextCount;
            witem->listCount = wptr->listCount;
            if (wptr->itemList != NULL)
                witem->itemList = wptr->itemList->itemInstanceCopy();
            else
                witem->itemList = NULL;
            if (wptr->loopList != NULL)
                witem->loopList = wptr->loopList->copyClass();
            else
                witem->loopList = NULL;
            wtail->next = witem;
            wptr = wptr->next;
        }
    }
    else {
        whead = witem;
        witem->next = NULL;
    }

    return whead;
}

uint8_t matchItemLists(ItemUtilities *itemUtilities, ItemInstance *list2) {

    if (list2 == NULL) return 0;
    if (list2->getGenericType() != genericType) return 0;

    if (genericType == GNRC_ITEM &&

```

```

        !itemUtilities->sameAs(list2->getItem(), itemid))
    return 0;
if (genericType == GNRC_ITEMLIST &&
    !(itemList->matchItemLists(itemUtilities, list2->getItemList()))
    return 0;

if (next == NULL && list2->getNext() == NULL) return 1;
if (next == NULL) return 0;
return next->matchItemLists(itemUtilities, list2->getNext());
}

uint8_t ItemInstance::matchLoopLists(ItemUtilities *itemUtilities,
                                     listT<ItemInstance> *list2) {

    void *wptr1, *wptr2;
    ItemInstance *witem1, *witem2;

    wptr1 = loopList->getFront();
    wptr2 = list2->getFront();

    if (wptr1 == NULL || wptr2 == NULL) return 0;

    while (1) {
        witem1 = loopList->lvalue(wptr1);
        witem2 = list2->lvalue(wptr2);

        if (witem1 == NULL || witem2 == NULL) return 0;
        if (witem1->getGenericType() != witem2->getGenericType()) return 0;

        if (witem1->getGenericType() == GNRC_ITEM &&
            witem1->getItem() != witem2->getItem())
            return 0;
        else if (witem1->getListID() != 0 && witem2->getListID() != 0) { //loop of 1
            printf("**ItemInstance**  matching loop of loops\n");
            return witem1->matchLoopLists(itemUtilities, witem2->getLoopList());
        }
        else if (witem1->getListID() != 0 || witem2->getListID() != 0) return 0;
        else if (witem1->getGenericType() == GNRC_ITEMLIST &&
            !(witem1->getItemList()->matchItemLists(itemUtilities,
                                                    witem2->getItemList()))
            return 0;

        wptr1 = loopList->revIterate(wptr1);

```

```

        wptr2 = list2->revIterate(wptr2);

        if (wptr1 == NULL && wptr2 == NULL) return 1;
        if (wptr1 == NULL || wptr2 == NULL) return 0;
    }

    return 1; // won't ever get here
}

unsigned char ItemInstance::setNext(ItemInstance *witem) {
    next = witem;
    witem->prev = this;
    nextCount++; // now upchannel
    return 1;
}

unsigned char ItemInstance::exists(unsigned int witem) {
    ItemInstance *wptr;
    wptr = this;

    while (wptr != NULL) {
        if (wptr->itemid == witem) return 1;
        wptr = wptr->next;
    }
    return 0;
}

};

```

7.5 Print

Header.

```
#pragma once
#include "Parameters.h"
#include "Terms.h"
#include "Patterns.h"
#include "Generics.h"
#include "TermList.h"
#include "Strings.h"
#include "utilities.h"
#include "ItemUtilities.h"
#include "TimelineEntry.h"

class Print
{
    ItemUtilities *itemUtilities;
    Utilities *utilities;
    Parameters *parameters;
    Terms *terms;
    Patterns *patterns;
    Generics *generics;
    Strings *strings;

    sqlite3_stmt *stmt_selItem;
    sqlite3_stmt *stmt_selTermNames;
    sqlite3_stmt *stmt_selActivity;
    sqlite3_stmt *stmt_selStringCount;

    char **sourceNames;
    uint32_t sourceCount;

    void indent(FILE *outputFile, uint8_t windent);
    void prtTimelineHeading(FILE *outputFile, uint8_t windent, uint32_t timelineid,
        uint32_t stime, uint32_t etime, uint32_t sourceid);
    void prtRecord(FILE *outputFile, uint8_t windent, TimelineEntry *entry);
    void prtTermDetail(FILE *outputFile, uint8_t windent, char *actorname,
        char *actionname, char *appname, char *wtypename, uint32_t witemid);
    void prtStringItem(FILE *outputFile, uint8_t windent, ItemInstance *itemlist);
    void prtLoopItems(FILE *outputFile, uint32_t windent, listT<ItemInstance> *loopList);

public:
    typedef struct replace_type {
```

```

        char *caseNbr;
        uint32_t startTime;
        uint32_t endTime;
        uint32_t timelineid;
        int term;
    } replace_t;

Print(Utilities *wutilities, ItemUtilities *witemUtilities,
      Parameters *wparameters, Patterns *wpatterns, Generics *wgenerics,
      Terms *wterms, Strings *wstrings);
~Print();
void prtStrings();
void prtTimelines();
void prtTimelines(listT<replace_t> *excludes);

uint8_t prtString(FILE *outputFile, TermList *wterms, uint32_t sid);
uint8_t prtTerm(FILE *outputFile, uint8_t windent, int wtermid, uint8_t printTerms
};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                      //
// Class: Print                                                                    //
// Input: date and case values from parameter table                               //
// Output: the events that match (and all terms and strings)                       //
// Notes:                                                                                      //
// Initial Release: 29-Sep-17                                                                //
//                                                                                      //
// Revision History:                                                                                      //
//                                                                                      //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "Print.h"
#include "Parameters.h"
#include "Terms.h"
#include "TimelineEntry.h"
#include "Timeline.h"
#include "Strings.h"
#include "TermList.h"

```

```

Print::Print(Utilities *wutilities, ItemUtilities *witemUtilities,
            Parameters *wparameters, Patterns *wpatterns,
            Generics *wgenerics, Terms *wterms, Strings *wstrings)

```

```

{
    char exec_string[2000];
    sqlite3_stmt *stmt;
    int rc;

    itemUtilities = witemUtilities;
    utilities = wutilities;
    parameters = wparameters;
    patterns = wpatterns;
    generics = wgenerics;
    terms = wterms;
    strings = wstrings;

    // Activity Reporting
    sprintf_s(exec_string, 2000, "SELECT activities.actorid, activities.actionid, acti
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selActivity, 0);
    //    if (rc != SQLITE_OK) { printf("Error: %i. couldn't prepare Select Activity Item

    sprintf_s(exec_string, 2000, "SELECT actors.description, actions.description, appe
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selTermNames, 0);
    if (rc != SQLITE_OK) { printf("Error: %i. couldn't prepare Select Terms Statement\

    // PrtString
    sprintf_s(exec_string, 2000, "SELECT count(*) FROM stringInstances WHERE stringID
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selStringCount, 0
    //    if (rc != SQLITE_OK) { printf("Error: %i. couldn't prepare Select Count of Strin

    // Source Descriptions
    sprintf_s(exec_string, 2000, "SELECT max(sourceid) FROM sources;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    //    if (rc != SQLITE_OK) { printf("Error: %i. couldn't prepare Select Source Stateme
    if (rc == SQLITE_OK) {
        sqlite3_reset(stmt);
        rc = sqlite3_step(stmt);
        sourceCount = (rc == SQLITE_ROW ? sqlite3_column_int(stmt, 0) : 0);
        sqlite3_finalize(stmt);
    }
    else
        sourceCount = 0;

```

```

if (sourceCount > 0) {
    sourceNames = (char **)malloc(sizeof(char *) * (sourceCount + 1));

    sprintf_s(exec_string, 2000, "SELECT sourceid, description FROM sources;");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare Select Source Statement\n", rc);
        return;
    }
    sqlite3_reset(stmt);
    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        sourceNames[sqlite3_column_int(stmt, 0)] =
            (char *)malloc(sizeof(char) *
                (strlen((char *)sqlite3_column_text(stmt, 1)) + 2));
        strcpy_s(sourceNames[sqlite3_column_int(stmt, 0)],
            strlen((char *)sqlite3_column_text(stmt, 1)) + 1,
            (char *)sqlite3_column_text(stmt, 1));
    }
} else
    sourceNames = NULL;

sqlite3_finalize(stmt);
}

Print::~Print()
{
    if (sourceNames != NULL) free(sourceNames);
}

void Print::prtStrings() {

    char showtime[80];
    time_t wtime;
    struct tm timeinfo;
    int wtermid;
    uint8_t isString;
    int rc;
    TermList *wterms;

    time(&wtime);
    localtime_s(&timeinfo, &wtime);

```

```

    strftime(showtime, 80, "%c", &timeinfo);
    fprintf(parameters->outFile, "\nReport Date: %s \n\n\n", showtime);

    terms->initialize();
    while ((wtermid = terms->iterate()) != -1) {
        isString = terms->isString(wtermid);

        if (isString && !parameters->showDetails) {
            wterms = strings->getStringTerms(wtermid, NULL, NULL);
            prtString(parameters->outFile, wterms, wtermid);
            delete wterms;
        } else {
            rc = 0;
            if (!isString && parameters->showTerms)
                fprintf(parameters->outFile, "Term %u\t", wtermid);
            if ((isString && parameters->showStrings) ||
                (!isString && parameters->showTerms)) {
                rc = prtTerm(parameters->outFile, 0, wtermid, 1);
            }
            if (rc) fprintf(parameters->outFile, "\n");
        }
    }
}

void Print::prtTimelines() {prtTimelines(NULL);}
void Print::prtTimelines(listT<replace_t> *excludes) {
    TimelineEntry *entry;
    Timeline *timeline;
    char *scaseNbr;
    uint32_t ctr, last_seen;
    time_t wtime;
    struct tm timeinfo;
    char showtime[82];
    void *liter, *prevIter;
    replace_t *nextReplace;

    time(&wtime);
    localtime_s(&timeinfo, &wtime);
    strftime(showtime, 80, "%c", &timeinfo);
    fprintf(parameters->outFile, "\nReport Date: %s          Abstraction Level: %u\n",
            showtime, parameters->existingLevel);

    timeline = new Timeline(utilities,parameters, terms, generics);

```


[illegible]

```

        delete entry;
    }
} while (scaseNbr = timeline->iterate());
}

void Print::indent(FILE *outputFile, uint8_t windent) {
    for (int i = 0; i < windent; i++)
        if (outputFile != NULL)
            fprintf(outputFile, "\t");
        else
            printf("\t");
}

// Print the timeline heading
void Print::prtTimelineHeading(FILE *outputFile, uint8_t windent,
                               uint32_t timelineid, uint32_t stime,
                               uint32_t etime, uint32_t sourceid) {
    char sstime[32], setime[32];
    time_t wtime;
    struct tm timeinfo;

    indent(outputFile, windent); // indent the correct number of tabs

    // print start time, end time, timelineid, and source name
    wtime = (time_t)(stime);
    localtime_s(&timeinfo, &wtime);
    strftime(ssstime, 30, "%c", &timeinfo);
    wtime = (time_t)(etime);
    localtime_s(&timeinfo, &wtime);
    strftime(setime, 30, "%c", &timeinfo);
    if (timelineid != 0)
        fprintf(outputFile, "%30s (%u) \t%30s (%u) \t%30u\t(source = %15s)",
                sstime, stime, setime, etime, timelineid,
                (sourceNames != NULL ? sourceNames[sourceid] : " "));
    else
        fprintf(outputFile, "%30s (%u) \t%30s (%u) \t%30s\t(source = %15s)",
                sstime, stime, setime, etime, "REPLACEMENT",
                (sourceNames != NULL ? sourceNames[sourceid] : " "));
}

uint8_t Print::prtTerm(FILE *outputFile, uint8_t windent, int wtermid,
                      uint8_t printTerms) {
    int rc;

```

```

uint8_t isString;
TermList *wtermList, *wtermPtr;

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "prtTerm -- term = %u\n", wtermid);
if (wtermid < 0 || wtermid > terms->getMaxTerm()) return 0;

isString = terms->isString(wtermid);
if (!isString) {
    if (printTerms) {
        sqlite3_reset(stmt_selTermNames);
        rc = sqlite3_bind_int(stmt_selTermNames, 1, wtermid);
        rc = sqlite3_step(stmt_selTermNames);
        printf("rc = %i\n", rc);
        if (rc == SQLITE_ROW)
            prtTermDetail(outputFile, windent,
                (char *)sqlite3_column_text(stmt_selTermNames, 0),
                (char *)sqlite3_column_text(stmt_selTermNames, 1),
                (char *)sqlite3_column_text(stmt_selTermNames, 2),
                (char *)sqlite3_column_text(stmt_selTermNames, 3),
                sqlite3_column_int(stmt_selTermNames, 4)); // actor, action, appexec,
    }
    return 1;
}

indent(outputFile, windent);
wtermList = strings->getStringTerms(wtermid, NULL, NULL);
if (wtermList == NULL) return 0;
rc = prtString(outputFile, wtermList, wtermid);
if (rc) {
    wtermPtr = wtermList;
    while (wtermPtr != NULL) {
        if (wtermPtr->getTerm() < 0 ||
            wtermPtr->getTerm() > terms->getMaxTerm()) return 0;
        prtTerm(outputFile, windent + 1, wtermPtr->getTerm(), 0);
        wtermPtr = wtermPtr->getNext();
    }
}

if (wtermList != NULL) delete wtermList;

return rc;
}

```

```

void Print::prtTermDetail(FILE *outputFile, uint8_t windent, char *actorname,
                        char *actionname, char *appname, char *wtypename,
                        uint32_t witemid) {
    char *itemName;

    indent(outputFile, windent);
    itemName = itemUtilities->getItemName(generics, witemid);
    fprintf(outputFile, "%15s\t%20s\t%15s\t\t%30s (%s)\n",
            actorname, actionname, appname, itemName, wtypename);
    if (itemName != NULL) free(itemName);
}

void Print::prtStringItem(FILE *outputFile, uint8_t windent, ItemInstance *itemList)
    char *itemName;
    ItemInstance *witem;

    witem = itemList;
    while (witem != NULL) {
        indent(outputFile, windent);
        if (witem->getGenericType() == GNRC_ITEM) {
            itemName = itemUtilities->getItemName(generics, witem->getItem());
            fprintf(outputFile, "%s\n", itemName);
            if (itemName != NULL) free(itemName);
        } else if (witem->getGenericType() == GNRC_ITEMLIST) { // should be the rest
            prtLoopItems(outputFile, windent + 2, witem->getLoopList());
        }
        witem = witem->getNext();
    }
}

void Print::prtLoopItems(FILE *outputFile, uint32_t windent,
                        listT<ItemInstance> *loopList) {
    void *liter;
    ItemInstance *witem;
    char *itemName;
    bool onlyOne;

    onlyOne = false;

    for (liter = loopList->getFront(); liter != NULL; liter = loopList->revIterate(liter))
        if (liter == loopList->getFront() &&
            loopList->getCount() > parameters->maxItems && parameters->maxItems > 0) {

```

```

        indent(outputFile, windent++);
        fprintf(outputFile, "%u items... sample:\n", loopList->getCount());
        onlyOne = true;
    }
    witem = loopList->lvalue(liter);
    if (witem->getGenericType() == GNRC_ITEMLIST)
        prtStringItem(outputFile, windent, witem->getItemList());
    else {
        indent(outputFile, windent + 2);
        itemName = itemUtilities->getItemName(generics, witem->getItem());
        fprintf(outputFile, "%s\n", itemName);
        if (itemName != NULL) free(itemName);
    }
    if (onlyOne) break;
}

}

void Print::prtRecord(FILE *outputFile, uint8_t windent, TimelineEntry *entry) {

    int rc;
    uint8_t foundTerm;
    char *itemName;
    listT<ItemInstance> *loopList;
    ItemInstance *witem;

    if (entry->getSourceID() == S_LOOP) {
        loopList = entry->getItems()->getLoopList();
        witem = loopList->lvalue(loopList->getFront());
        if (witem->getGenericType() == GNRC_ITEMLIST)
            fprintf(outputFile, " (Loop %u   String %i)\n",
                    entry->getActivity(), entry->getTerm());
        else
            fprintf(outputFile, " (Loop %u   Term %i)\n",
                    entry->getActivity(), entry->getTerm());
        if (parameters->showDetails)
            prtTerm(outputFile, windent, entry->getTerm(), 1); //termid
        if (parameters->showItems) {
            windent += 2;
            prtLoopItems(outputFile, windent, loopList);
        }
    } else if (entry->getSourceID() == S_STRING) {
        fprintf(outputFile, " (Instance %u   String %i)\n",

```

```

        entry->getActivity(), entry->getTerm());
if (parameters->showDetails)
    prtTerm(outputFile, windent, entry->getTerm(), 1);
if (parameters->showItems) {
    windent += 2;
    if (entry->getItems()->getGenericType() == GNRC_ITEMLIST)
        prtStringItem(outputFile, windent,
            entry->getItems()->getItemList()); //activityid
    else
        printf("Instance %i doesn't have item list\n",
            entry->getActivity());
}
}
else {
    if (entry->getTerm() > 0) {
        fprintf(outputFile, " (Activity %u   Term %i)\n",
            entry->getActivity(), entry->getTerm());
        foundTerm = 1;
    }
    else {
        fprintf(outputFile, " (Activity %u)\n", entry->getActivity());
        foundTerm = 0;
    }
    if (!foundTerm || parameters->showDetails) {
        sqlite3_bind_int(stmt_selActivity, 1, entry->getActivity());
        rc = sqlite3_step(stmt_selActivity);
        if (rc == SQLITE_ROW)
            prtTermDetail(outputFile, windent,
                (char *)sqlite3_column_text(stmt_selActivity, 5),
                (char *)sqlite3_column_text(stmt_selActivity, 6),
                (char *)sqlite3_column_text(stmt_selActivity, 7),
                (char *)sqlite3_column_text(stmt_selActivity, 8),
                sqlite3_column_int(stmt_selActivity, 4)); // actor, action, appexec
    } else if (!parameters->showDetails) {
        indent(outputFile, windent + 2);
        itemName = itemUtilities->getItemName(generics, entry->getItems()->getItem());
        fprintf(outputFile, "%s\n", itemName);
        if (itemName != NULL) free(itemName);
    }
}
}
}

```

```

uint8_t Print::prtString(FILE *outputFile, TermList *wterms, uint32_t sid) {

    uint32_t lastActor, lastAction, lastAppexec, lastItemtype, lastItem;
    uint32_t wactor, waction, wappexec, witemtypeid, witemid;
    char pactorName[1000], pactionName[1000], pappexecName[1000], ptypeName[1000];
    char *genericName, *witemname;
    uint32_t dupActors, dupActions, dupAppexecs, dupItems;

    int rc, wcount;
    uint8_t isString, isGeneric, notFirst, firstPrint;
    char *sname;

    if (parameters->debugParms >= 3)
        fprintf(parameters->logFile, "**prtString -- sid = %i  (%llx)\n",
                sid, (uint64_t)wterms);
    // Return if only show strings with counts
    wcount = 0;
    if (parameters->showCounts && sid != 0) {
        sqlite3_reset(stmt_selStringCount);
        sqlite3_bind_int(stmt_selStringCount, 1, sid);
        rc = sqlite3_step(stmt_selStringCount);
        wcount = (rc == SQLITE_ROW ? sqlite3_column_int(stmt_selStringCount, 0) : 0);
        if (wcount == 0 && !(parameters->showZero)) return 0;
    }

    if (sid != 0) {
        sname = strings->getName(sid);
        if (sname == NULL || strlen(sname) < 2) {
            sname = (char *)malloc(sizeof(char) * 15);
            sprintf_s(sname, 14, "String %6u", sid);
        }
        fprintf(outputFile, "%s", sname);
        free(sname);
    }
    else
        fprintf(outputFile, "String Description:\n");

    if (parameters->showCounts) {
        fprintf(outputFile, "\t\t(Count = %6i)", wcount);
    }
    fprintf(outputFile, "\t\t\t");

    lastItem = lastItemtype = lastActor = lastAction = lastAppexec = 0;

```

```

dupActors = dupActions = dupAppexecs = dupItems = 0;
notFirst = 0;
firstPrint = 1;
sprintf_s(pactorName, 998, " ");
sprintf_s(pactionName, 998, " ");
sprintf_s(pappexecName, 998, " ");
sprintf_s(ptypeName, 998, " ");

while (1) {
    if (wterms != NULL) {
        if (wterms->getTerm() < 0 || wterms->getTerm() > terms->getMaxTerm())
            break;
        sqlite3_reset(stmt_selTermNames);
        sqlite3_bind_int(stmt_selTermNames, 1, wterms->getTerm());
        rc = sqlite3_step(stmt_selTermNames);
        if (rc != SQLITE_ROW) { wterms = wterms->getNext(); continue; }
        wactor = terms->getActor(wterms);
        waction = terms->getAction(wterms);
        wappexec = terms->getAppexec(wterms);
        witemtypeid = terms->getItemtype(wterms);
        witemid = terms->getItem(wterms);
        isString = terms->isString(wterms);
    }

    // If the item has changed, print what you've got
    if ((wterms == NULL && lastItem != 0) ||
        ((lastItem != 0) && (lastItem != witemid || lastItemtype != witemtypeid
            || isString == 1))) {
        if (!firstPrint) fprintf(outputFile, " + ");
        firstPrint = 0;

        // Print the Actor, Application, and Action
        fprintf(outputFile, "(");
        if (dupActors < dupItems)
            fprintf(outputFile, "(Actors %s) ", pactorName);
        else
            fprintf(outputFile, "Actor %s, ", pactorName);
        if (dupAppexecs < dupItems)
            fprintf(outputFile, "(Applications %s) ", pappexecName);
        else
            fprintf(outputFile, "Application %s, ", pappexecName);
        if (dupActions < dupItems)
            fprintf(outputFile, "(%s) ", pactionName);
    }
}

```



```

else
    fprintf(outputFile, "%s", pactionName);
fprintf(outputFile, ")\n");

// Print the item (if it's generic, you need the item type name from the term)
isGeneric = (rc == SQLITE_ROW ?
    (itemUtilities->getItemtype(lastItem) == I_GENERIC ? 1 : 0) : 0);
if (isGeneric) {
    genericName = itemUtilities->getItemName(generics, lastItem);
    witemname = (char *)malloc(sizeof(char) *
        (strlen(genericName) + strlen(ptypeName) + 3));
    sprintf_s(witemname, strlen(genericName) + strlen(ptypeName) + 2,
        "%s %s", ptypeName, genericName);
    fprintf(outputFile, "(%s)", witemname);
    free(genericName);
    free(witemname);
}
else {
    witemname = itemUtilities->getItemName(generics, lastItem);
    fprintf(outputFile, "(%s %s)", ptypeName, witemname);
    free(witemname);
}

// If the exact same thing showed up multiple times, just put a xN
// instead of printing it N times
if ((dupActors == dupItems) && (dupActions == dupItems) &&
    (dupAppexecs == dupItems) && (dupItems > 0))
    fprintf(outputFile, " x%u", dupItems + 1);

notFirst = 0;
}
if (wterms == NULL) break;

// If the term is a string, print the string #
if (isString) {
    if (!firstPrint) fprintf(outputFile, " + ");
    firstPrint = 0;
    fprintf(outputFile, "String %i", wterms->getTerm());
    sprintf_s(pactorName, 998, " ");
    sprintf_s(pactionName, 998, " ");
    sprintf_s(pappexecName, 998, " ");
    sprintf_s(ptypeName, 998, " ");
    dupActors = dupActions = dupAppexecs = dupItems = 0;
}

```

```

    notFirst = 0;
}

// Otherwise, it's an item
else {
    // Make a note of the term's typename
    sprintf_s(ptypeName, 998, "%s", // Item Type Description
        (char *)sqlite3_column_text(stmt_selTermNames, 3));

    // Check if the actor, action or application occurs multiple times
    // If so, make a note of it; if not, add the new name
    if (notFirst) {
        if (lastActor == wactor)
            dupActors++;
        else
            sprintf_s(pactorName, 998, "%s + %s", pactorName,
                (char *)sqlite3_column_text(stmt_selTermNames, 0));
        if (lastAction == waction)
            dupActions++;
        else
            sprintf_s(pactionName, 998, "%s + %s", pactionName,
                (char *)sqlite3_column_text(stmt_selTermNames, 1));
        if (lastAppexec == wappexec)
            dupAppexecs++;
        else
            sprintf_s(pappexecName, 998, "%s + %s", pappexecName,
                (char *)sqlite3_column_text(stmt_selTermNames, 2));
        dupItems++;
    }
    // save then name of the actor, action, and application
    else {
        sprintf_s(pactorName, 998, "%s",
            (char *)sqlite3_column_text(stmt_selTermNames, 0));
        sprintf_s(pactionName, 998, "%s",
            (char *)sqlite3_column_text(stmt_selTermNames, 1));
        sprintf_s(pappexecName, 998, "%s",
            (char *)sqlite3_column_text(stmt_selTermNames, 2));
        dupActors = dupActions = dupAppexecs = dupItems = 0;
        notFirst = 1;
    }
}

lastItemtype = witemtypeid;
lastItem = witemid;

```

```
        lastActor = wactor;  
        lastAction = waction;  
        lastAppexec = wappexec;  
  
        wterms = wterms->getNext();  
    }  
    fprintf(outputFile, "\n");  
  
    return 1;  
}
```

7.6 Search

Header.

```
#pragma once
#include <unordered_map>
#include "utilities.h"
#include "TearCPP.h"
#include "Parameters.h"
#include "Terms.h"
#include "Patterns.h"
#include "Generics.h"
#include "Strings.h"
#include "Print.h"
#include "Timeline.h"

#define htSearch_i std::unordered_map<uint32_t, searches *>::const_iterator
#define htInstance_i std::unordered_map<uint32_t, instance_entry >::const_iterator

class Search
{
    typedef struct search_entry_type {
        unsigned int term;
        unsigned int instance;
        struct search_entry_type *next;
    } search_entry;

    typedef struct searchInstances_type {
        unsigned int searchId;
        unsigned int startTime; // when this instance started in the timeline
        unsigned int endTime;   // once the entire instance matches, when this
                                // instance ended in the timeline

        char *caseNbr;
        search_entry *nextMatch;
        listT<uint32_t> *timelines;
        struct searchInstances_type *prev;
        struct searchInstances_type *next;
    } search_instances_entry;

    typedef struct searches_type {
        unsigned int id;
        int replacement;
        search_entry *head;
        search_entry *tail;
    }
```

```

    search_instances_entry *openInstances;
    search_instances_entry *openInstancesTail;
    search_instances_entry *closedInstances;
    search_instances_entry *closedInstancesTail;
} searches;

typedef struct instance_type {
    uint32_t id;
    uint32_t genericType;
    char * genericValue;
} instance_entry;

ItemUtilities *itemUtilities;
Utilities *utilities;
Parameters *parameters;
Terms *terms;
Patterns *patterns;
Generics *generics;
Strings *strings;
Timeline *timeline;

sqlite3_stmt *stmt_selActivity;

std::unordered_map<uint32_t, searches *> htSearches;
std::unordered_map<uint32_t, instance_entry > htInstances;

static bool dateRangeLessThan(const Parameters::dateRange_t *lhs,
                              const Parameters::dateRange_t *rhs) {
    if (strcmp(lhs->caseNbr, rhs->caseNbr) < 0) { return true; }
    if (strcmp(lhs->caseNbr, rhs->caseNbr) > 0) { return false; }

    // cases equal
    if (lhs->startDate < rhs->startDate) { return true; }
    if (lhs->startDate > rhs->startDate) { return false; }

    // startDates equal
    if (lhs->endDate < rhs->endDate) { return true; }
    if (lhs->endDate > rhs->endDate) { return false; }

    // false if elements are equal
    return false;
}

```

```

static bool replaceLT(const Print::replace_t *lhs,
                     const Print::replace_t *rhs) {
    if (strcmp(lhs->caseNbr, rhs->caseNbr) < 0) return true;
    if (strcmp(lhs->caseNbr, rhs->caseNbr) > 0) return false;

    // cases equal
    if (lhs->startTime < rhs->startTime) { return true; }
    if (lhs->startTime > rhs->startTime) { return false; }

    // times equal
    if (lhs->timelineid < rhs->timelineid) { return true; }
    if (lhs->timelineid > rhs->timelineid) { return false; }

    // false if elements are equal
    return false;
}

void checkSearches(TimelineEntry *entry);
bool checkTerm(searches *wsearch, search_instances_entry *wsearchInstance,
               uint32_t endTime, char *scaseNbr,
               uint32_t wtermid, ItemInstance *witems, uint32_t wtimelineid);
bool checkTermDetails(searches *wsearch,
                      search_instances_entry *wsearchInstance, uint32_t endTime,
                      char *scaseNbr, uint32_t wactor, uint32_t waction,
                      uint32_t wappexec, uint32_t witemtype, ItemInstance *witems,
                      uint32_t wtimelineid);
void insertRange(listT<Parameters::dateRange_t> *head, uint32_t startTime,
                uint32_t endTime, char *scaseNbr);
bool nextMatch(searches *wsearch, search_instances_entry *wsearchInstance,
               uint32_t endTime, char *scaseNbr);

public:
    Search(Utilities *wutilities, ItemUtilities *witemUtilities,
           Parameters *wparameters, Patterns *wpatterns,
           Generics *wgenerics, Terms *wterms, Strings *wstrings);
    ~Search();
    void searchTimeline();
};

```

Header.

```

////////////////////////////////////
//
// Class: Search
//

```

```

// Input: date and search terms from parameter table //
// Output: the events that match -- fed to PrtTimeline //
// Notes: //
// Initial Release: 29-Sep-17 //
// //
// Revision History: //
// //
////////////////////////////////////
#include "stdafx.h"
#include "Search.h"
#include "Timeline.h"
#include "TimelineEntry.h"
#include "Print.h"

Search::Search(Utilities *wutilities, ItemUtilities *witemUtilities,
               Parameters *wparameters, Patterns *wpatterns, Generics *wgenerics,
               Terms *wterms, Strings *wstrings) {
    htSearch_i htiter;

    instance_entry winstance;
    char exec_string[2000];
    int rc;
    FILE *parmFile;
    searches *wsearch;
    TermList *wtermList;
    search_entry *wsearch_entry;
    char *winput;
    uint32_t wid, witerm, wiinstance;
    uint8_t isInstances, isSearches, isReplacements;
    int wireplacement;

    utilities = wutilities;
    parameters = wparameters;
    strings = wstrings;
    terms = wterms;
    patterns = wpatterns;
    generics = wgenerics;
    timeline = new Timeline(utilities, parameters, terms, generics);

    sprintf_s(exec_string, 2000, "SELECT actorid, actionid, applicationid, items.itemt
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt_selActivity, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Activity Statement\n", rc);

```

```

    return;
}

fopen_s(&parmFile, parameters->parmFilename, "r");
if (parmFile == NULL) {
    printf("Could not open parameter file %s\n", parameters->parmFilename);
    return;
}

// Load the parameters from the parameter file
isInstances = 0;
isSearches = 0;
isReplacements = 0;
while (!feof(parmFile)) {
    winput = utilities->get_cfield(parmFile);

    // skip comments
    if ((strlen(winput) > 2) && (winput[0] == '/') && (winput[1] == '/')) {
        free(winput); winput = NULL;
        utilities->getEoln(parmFile);
        continue;
    }

    if (parameters->isKeyword(winput) &&
        utilities->strcmpU("instances", winput) &&
        utilities->strcmpU("searches", winput) &&
        utilities->strcmpU("replacements", winput)) {
        isInstances = 0;
        isSearches = 0;
        isReplacements = 0;
    } else if (!utilities->strcmpU("instances", winput)) {
        isInstances = 1;
        isSearches = 0;
        isReplacements = 0;
    } else if (!utilities->strcmpU("searches", winput)) {
        isInstances = 0;
        isSearches = 1;
        isReplacements = 0;
    } else if (!utilities->strcmpU("replacements", winput)) {
        isInstances = 0;
        isSearches = 0;
        isReplacements = 1;
    } else if (isInstances) {

```



```

winstance.id = atoi(wininput);
if (winstance.id == 0) {
    free(wininput); wininput = NULL;
    utilities->getEoln(parmFile);
    continue;
}
winstance.genericType = utilities->get_ifield(parmFile);
winstance.genericValue = utilities->get_cfield(parmFile);
if (parameters->debugParms >= 3)
    printf("Instance id = %u\t type = %u\t   str = %s\n",
           winstance.id, winstance.genericType, winstance.genericValue);
htInstances[winstance.id] = winstance;
} else if (isReplacements) {
    wid = atoi(wininput);
    if (wid == 0) {
        free(wininput); wininput = NULL;
        utilities->getEoln(parmFile);
        continue;
    }
    wireplacement = utilities->get_ifield(parmFile);
    if (htiter == htSearches.end()) {
        wsearch = (searches *)malloc(sizeof(searches) * 1);
        wsearch->id = wid;
        wsearch->head = NULL;
        wsearch->tail = NULL;
        wsearch->openInstances = NULL;
        wsearch->closedInstances = NULL;
        wsearch->closedInstancesTail = NULL;
        htSearches[wid] = wsearch;
        if (parameters->debugParms >= 1)
            printf("added search %u  \n", wid);
    } else
        wsearch = htiter->second;
    wsearch->replacement = wireplacement;
    if (parameters->debugParms >= 1)
        printf("added replacement %i to search %u  \n", wireplacement, wid);
} else if (isSearches) {
    wid = atoi(wininput);
    if (wid == 0) {
        free(wininput); wininput = NULL;
        utilities->getEoln(parmFile);
        continue;
    }
}

```

```

witerm = utilities->get_ifield(parmFile);
wiinstance = utilities->get_ifield(parmFile);
htiter = htSearches.find(wid);
if (htiter == htSearches.end()) {
    wsearch = (searches *)malloc(sizeof(searches) * 1);
    wsearch->id = wid;
    wsearch->head = NULL;
    wsearch->tail = NULL;
    wsearch->openInstances = NULL;
    wsearch->closedInstances = NULL;
    wsearch->closedInstancesTail = NULL;
    htSearches[wid] = wsearch;
    if (parameters->debugParms >= 1) printf("added search %u \n", wid);
}
// get the item associated with the current term -- the loop is
// there to let strings get expanded out to terms
wtermList = new TermList();
wtermList->setTerm(witerm);
wtermList = strings->flatten(wtermList, NULL);
while (wtermList != NULL) {
    wsearch_entry = (search_entry *)malloc(sizeof(search_entry) * 1);
    wsearch_entry->term = wtermList->getTerm();
    wsearch_entry->instance = wiinstance;
    wsearch_entry->next = NULL;
    if (wsearch->head == NULL) wsearch->head = wsearch_entry;
    if (wsearch->tail != NULL) {
        (wsearch->tail)->next = wsearch_entry;
    }
    wsearch->tail = wsearch_entry;
    if (parameters->debugParms >= 1)
        printf("\tadded search id %u\t term entry\t\tTerm = %u\tInstance = %u\n",
            wsearch->id, wsearch_entry->term, wsearch_entry->instance);
    wtermList = wtermList->getNext();
}
delete wtermList;
}

else { isInstances = 0; isSearches = 0; }
free(winput); winput = NULL;
}

fclose(parmFile);

```

```
}
```

```
Search::~Search()
```

```
{
```

```
}
```

```
void Search::searchTimeline() {
```

```
    TimelineEntry *entry;
```

```
    htSearch_i htiter;
```

```
    searches *wsearches;
```

```
    search_instances_entry *wsearchInstance, *wsientry;
```

```
    listT<Parameters::dateRange_t> *newDateRange;
```

```
    time_t wtimer;
```

```
    struct tm timeinfo;
```

```
    char stime[80], etime[80];
```

```
    char *scaseNbr;
```

```
    listT<Print::replace_t> *replacements;
```

```
    Print::replace_t *nextReplace;
```

```
    void *liter;
```

```
    if ((scaseNbr = timeline->initialize()) == NULL) return;
```

```
    replacements = new listT<Print::replace_t>;
```

```
    do {
```

```
        printf("\tExamining timeline for relevant temporal blocks\n");
```

```
        while ((entry = timeline->process(0)) != NULL) {
```

```
            checkSearches(entry);
```

```
        }
```

```
    } while (scaseNbr = timeline->iterate());
```

```
    newDateRange = new listT<Parameters::dateRange_t>;
```

```
    for (htiter = htSearches.begin(); htiter != htSearches.end(); htiter++) {
```

```
        wsearches = htiter->second;
```

```
        wsearchInstance = wsearches->closedInstances;
```

```
        while (wsearchInstance != NULL) {
```

```
            if (wsearchInstance->endTime != 0) {
```

```
                wtimer = (time_t)wsearchInstance->startTime;
```

```
                localtime_s(&timeinfo, &wtimer);
```

```
                strftime(stime, 80, "%x %X", &timeinfo);
```

```

wtimer = (time_t)wsearchInstance->endTime;
localtime_s(&timeinfo, &wtimer);
strftime(etime, 80, "%x %X", &timeinfo);

printf("\tCase %s\t\t%s - %s\n", wsearchInstance->caseNbr, stime, etime);

// newDateRange -- sorted by Casenbr, startDate, endDate
insertRange(newDateRange, wsearchInstance->startTime - parameters->bufferTime,
            wsearchInstance->endTime + parameters->bufferTime,
            wsearchInstance->caseNbr);

if (wsearches->replacement != 0) {
    if (wsearchInstance->timelines->lempty()) {
        nextReplace = (Print::replace_t *)
            malloc(sizeof(Print::replace_t) * 1);
        nextReplace->term = wsearches->replacement;
        nextReplace->startTime = wsearchInstance->startTime;
        nextReplace->endTime = wsearchInstance->endTime;
        nextReplace->caseNbr = (char *)
            malloc(sizeof(char) * (strlen(wsearchInstance->caseNbr) + 2));
        strcpy_s(nextReplace->caseNbr,
            strlen(wsearchInstance->caseNbr) + 1,
            wsearchInstance->caseNbr);
        liter = wsearchInstance->timelines->getFront();
        nextReplace->timelineid = *(wsearchInstance->timelines->lvalue(liter)
            replacements->insertSort(nextReplace, &replaceLT);
        liter = wsearchInstance->timelines->revIterate(liter);
        while (liter != NULL) {
            nextReplace = (Print::replace_t *)
                malloc(sizeof(Print::replace_t) * 1);
            nextReplace->term = 0; // only 1st timelineid is replaced, rest
            nextReplace->startTime = wsearchInstance->startTime;
            nextReplace->endTime = wsearchInstance->endTime;
            nextReplace->caseNbr = (char *)malloc(sizeof(char) * (
                strlen(wsearchInstance->caseNbr) + 2));
            strcpy_s(nextReplace->caseNbr,
                strlen(wsearchInstance->caseNbr) + 1,
                wsearchInstance->caseNbr);
            nextReplace->timelineid = *(wsearchInstance->timelines->lvalue(liter)
            replacements->insertSort(nextReplace, &replaceLT);
            liter = wsearchInstance->timelines->revIterate(liter);
        }
    }
}

```

```

        }
    }
}
wsientry = wsearchInstance;
wsearchInstance = wsearchInstance->next;
if (wsientry->caseNbr != NULL) free(wsientry->caseNbr);
free(wsientry);
wsientry = NULL;
}
printf("\n\n");
}
htSearches.clear();

parameters->dateRange = newDateRange;
parameters->scaseList = NULL;

printf("Printing results...\n");
Print *action = new Print(utilities, itemUtilities, parameters, patterns,
                           generics, terms, strings);
if (replacements->lempty())
    action->prtTimelines();
else
    action->prtTimelines(replacements);
}

void Search::checkSearches(TimelineEntry *entry) {

    htSearch_i htiter;
    searches *wsearch;
    search_instances_entry *wsearchInstance, *wsiptr, *wsiTail;
    bool isDone;

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "entering checkSearches\n");

    for (htiter = htSearches.begin(); htiter != htSearches.end(); htiter++) {
        wsearch = htiter->second;
        wsearchInstance = wsearch->openInstances;
        wsiTail = NULL;
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "\tchecking search %i\n", wsearch->id);
        while (wsearchInstance != NULL) {
            if ((wsearchInstance->endTime == 0) &&

```

```

        (entry->getStartTime() > wsearchInstance->startTime +
         parameters->timespan)) {
    if (wsiTail != NULL)
        wsiTail->next = wsearchInstance->next;
    if (wsearch->openInstances == wsearchInstance)
        wsearch->openInstances = wsearchInstance->next;
    wsiptr = wsearchInstance;
    wsearchInstance = wsearchInstance->next;
    if (wsiptr->caseNbr != NULL) free(wsiptr->caseNbr);
    free(wsiptr);
    continue;
}

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
            "\t\tchecking instance %i (start = %u    end = %u)\n",
            wsearchInstance->searchId, wsearchInstance->startTime,
            wsearchInstance->endTime);
if (entry->getTerm() == 0)
    isDone = checkTermDetails(wsearch, wsearchInstance,
                             entry->getEndTime(), entry->getCaseNbr(),
                             entry->getActor(), entry->getAction(),
                             entry->getAppexec(), entry->getItemtype(),
                             entry->getItems(), entry->getTimelineID());
else
    isDone = checkTerm(wsearch, wsearchInstance, entry->getEndTime(),
                      entry->getCaseNbr(), entry->getTerm(),
                      entry->getItems(), entry->getTimelineID());

if (isDone) {
    if (wsiTail != NULL) wsiTail->next = wsearchInstance->next;
    if (wsearch->openInstances == wsearchInstance)
        wsearch->openInstances = wsearchInstance->next;
    wsearchInstance->next = NULL;
}
wsiTail = wsearchInstance;
wsearchInstance = wsearchInstance->next;
}

// Check if the first term matches -- if so, create a new search instance
wsearchInstance = (search_instances_entry *)malloc(sizeof(search_instances_entry));
wsearchInstance->searchId = wsearch->id;

```

```

wsearchInstance->caseNbr = (char *)malloc(sizeof(char) * (
                                strlen(entry->getCaseNbr()) + 2));
strcpy_s(wsearchInstance->caseNbr, strlen(entry->getCaseNbr()) + 1,
                                entry->getCaseNbr());
wsearchInstance->startTime = entry->getStartTime();
wsearchInstance->endTime = 0;
wsearchInstance->nextMatch = wsearch->head;
wsearchInstance->timelines = new listT<uint32_t>;
wsearchInstance->next = NULL;

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\t\tchecking new instance      id = %i      term
                                wsearch->head->instance, entry->getTerm());
if (entry->getTerm() == 0)
    isDone = checkTermDetails(wsearch, wsearchInstance, entry->getEndTime(),
                                entry->getCaseNbr(), entry->getActor(),
                                entry->getAction(), entry->getAppexec(),
                                entry->getItemtype(), entry->getItems(),
                                entry->getTimelineID());
else
    isDone = checkTerm(wsearch, wsearchInstance, entry->getEndTime(),
                                entry->getCaseNbr(), entry->getTerm(),
                                entry->getItems(), entry->getTimelineID());

if (wsearchInstance->nextMatch != wsearch->head) { // at least 1st term matched
    if (!isDone) {
        if (wsearch->openInstances != NULL)
            wsiTail->next = wsearchInstance;
        if (wsearch->openInstances == NULL)
            wsearch->openInstances = wsearchInstance;
    }
}
else {
    free(wsearchInstance->caseNbr);
    free(wsearchInstance);
}
}

}

bool Search::checkTerm(searches *wsearch, search_instances_entry *wsearchInstance,
                        uint32_t endTime, char *scaseNbr,
                        uint32_t wtermid, ItemInstance *witems,

```

```

        uint32_t wtimelineid) {
    TermList *wtermList;
    bool isDone;

    if (parameters->debugParms >= 5)
        fprintf(parameters->logFile, "\t\t\tcheckTerm\n");
    if (wtermid == 0 || witems == NULL) return false;

    if (terms->isString(wtermid)) {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile,
                    "\t\t\t\tchecking string details   sid = %i\n",wtermid);
        wtermList = strings->getStringTerms(wtermid,NULL,NULL);
        while (wtermList != NULL) {
            isDone = checkTerm(wsearch, wsearchInstance, endTime, scaseNbr,
                               wtermList->getTerm(), witems, wtimelineid);
            if (isDone) return true;
            wtermList = wtermList->getNext();
        }
    } else {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile,
                    "\t\t\t\tchecking term details       termid = %i\n",wtermid);
        isDone = checkTermDetails(wsearch, wsearchInstance, endTime, scaseNbr,
                                   terms->getActor(wtermid), terms->getAction(wtermid),
                                   terms->getAppexec(wtermid), terms->getItemtype(wtermid),
                                   witems, wtimelineid);

        return isDone;
    }
    return false;
}

bool Search::checkTermDetails(searches *wsearch,
                             search_instances_entry *wsearchInstance, uint32_t endTime,
                             char *scaseNbr, uint32_t wactor, uint32_t waction,
                             uint32_t wappexec, uint32_t witetype, ItemInstance *witems,
                             uint32_t wtimelineid) {
    htInstance_i htiter;
    uint32_t *timelineid;
    // Remember -- we're not trying to match everything -
    // a single match means we move forward
    uint32_t xactor, xaction, xappexec, xitetype;
    ItemInstance *witelist;

```



```

listT<ItemInstance> *wloopList;
instance_entry winstance;
void *wptr;
char *wname;
bool isDone;
search_entry *wnextMatch;

if (wsearchInstance != NULL)
    wnextMatch = wsearchInstance->nextMatch;
else
    wnextMatch = wsearch->head;

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile, "\t\t\tcheckTermDetails\n");
if (wnextMatch == NULL || witems == NULL) return false;

xactor = terms->getActor(wnextMatch->term);
xaction = terms->getAction(wnextMatch->term);
xappexec = terms->getAppexec(wnextMatch->term);
xitemtype = terms->getItemtype(wnextMatch->term);
htiter = htInstances.find(wnextMatch->instance);
if (htiter == htInstances.end()) return false;
winstance = htiter->second;

if (parameters->debugParms >= 5)
    fprintf(parameters->logFile,
            "\t\t\tactor = %i    action = %i    app = %i    type = %i    \n",
            xactor, xaction, xappexec, xitemtype);
if (xactor == wactor && xaction == waction && xappexec == wappexec &&
    xitemtype == witemtype) {
    if (witems->getGenericType() == GNRC_ITEM) {
        wname = itemUtilities->getItemName(generics, witems->getItem());
        utilities->strtoupperIP(wname);
        if (utilities->matchu(winstance.genericValue, wname) &&
            winstance.genericType == GNRC_ITEM) {
            if (parameters->debugParms >= 5)
                fprintf(parameters->logFile, "matched\n");
            free(wname);
            isDone = nextMatch(wsearch, wsearchInstance, endTime, scaseNbr);
            timelineid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
            *timelineid = wtimelineid;
            wsearchInstance->timelines->insertRear(timelineid);
            return isDone;
        }
    }
}

```

```

    }
    free(wname);
} else if (witems->getGenericType() == GNRC_ITEMLIST) { // only other choice ri
    if (witems->getListID() != 0) {
        wloopList = witems->getLoopList();
        wptr = wloopList->getFront();
        while (wptr != NULL) {
            wname = itemUtilities->getItemName(generics,
                                                wloopList->lvalue(wptr)->getItem());
            utilities->strtoupperIP(wname);
            if (utilities->matchu(winstance.genericValue, wname) &&
                winstance.genericType == GNRC_ITEM) {
                if (parameters->debugParms >= 5)
                    fprintf(parameters->logFile, "matched\n");
                isDone = nextMatch(wsearch, wsearchInstance, endTime,
                                    scaseNbr);
                free(wname);
                timelineid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
                *timelineid = wtimelineid;
                wsearchInstance->timelines->insertRear(timelineid);
                return isDone;
            }
            free(wname);
            wptr = wloopList->revIterate(wptr);
        }
        return false;
    }
} else {
    witemList = witems->getItemList();
    while (witemList != NULL) {
        wname = itemUtilities->getItemName(generics, witemList->getItem());
        utilities->strtoupperIP(wname);
        if (utilities->matchu(winstance.genericValue, wname) &&
            winstance.genericType == GNRC_ITEM) {
            if (parameters->debugParms >= 5)
                fprintf(parameters->logFile, "matched\n");
            isDone = nextMatch(wsearch, wsearchInstance, endTime, scaseNbr);
            free(wname);
            timelineid = (uint32_t *)malloc(sizeof(uint32_t) * 1);
            *timelineid = wtimelineid;
            wsearchInstance->timelines->insertRear(timelineid);
            return isDone;
        }
    }
}

```

```

        free(wname);
        witemList = witemList->getNext();
    }
}
return false;
}
return false;
}

```

```

bool Search::nextMatch(searches *wsearch, search_instances_entry *wsearchInstance,
    uint32_t endTime, char *scaseNbr) {

```

```

    if (wsearchInstance == NULL) {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "creating new search instance\n");
        wsearchInstance = (search_instances_entry *)
            malloc(sizeof(search_instances_entry) * 1);
        wsearchInstance->searchId = wsearch->id;
        wsearchInstance->caseNbr = (char *)
            malloc(sizeof(char) * (strlen(scaseNbr) + 2));
        strcpy_s(wsearchInstance->caseNbr, strlen(scaseNbr) + 1, scaseNbr);
        wsearchInstance->startTime = endTime;
        wsearchInstance->endTime = 0;
        wsearchInstance->timelines = new listT<uint32_t>;
        wsearchInstance->next = NULL;
        if (wsearch->openInstances != NULL)
            wsearchInstance->next = wsearch->openInstances;
        wsearch->openInstances = wsearchInstance;
        wsearchInstance->nextMatch = wsearch->head->next;
    }

```

```

    else {
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "moving to next search entry\n");
        wsearchInstance->nextMatch = (wsearchInstance->nextMatch->next);
    }

```

```

    if (wsearchInstance->nextMatch == NULL) { // end of the search - move to the close
        if (parameters->debugParms >= 5)
            fprintf(parameters->logFile, "completed search instance\n");
        wsearchInstance->endTime = endTime;
        if (wsearch->closedInstancesTail != NULL)

```

```

        wsearch->closedInstancesTail->next = wsearchInstance;
    if (wsearch->closedInstances == NULL)
        wsearch->closedInstances = wsearchInstance;
    wsearch->closedInstancesTail = wsearchInstance;
    return true;
}
return false;
}

void Search::insertRange(listT<Parameters::dateRange_t> *head, uint32_t startTime,
                        uint32_t endTime, char *scaseNbr) {
    void *liter;
    Parameters::dateRange_t *wptr, *wtail, *newRange;

    if (startTime > endTime) return; // poorly formed
                                    // Example: new date range = (50, 75)
                                    // Set up

    wptr = NULL;
    wtail = NULL;

    ////////////
    // Find the first date range where the start time is greater than the new one
    for (liter = head->getFront(); liter != NULL; liter = head->revIterate(liter)) {
        wptr = head->lvalue(liter);
        if (strcmp(wptr->caseNbr, scaseNbr) == 0) {
            if (wptr->startDate >= startTime) break;
            wtail = wptr;
        }
    }

    // the new date range starts after the last date range in the list starts
    if (wptr == NULL) {

        // Case 1: list = Null
        // Case 2: list = (10, 20), (30, 40)
        if (head->lempty() || wtail->endDate < startTime) { // end of list
            newRange = (Parameters::dateRange_t *)
                malloc(sizeof(Parameters::dateRange_t) * 1);
            newRange->startDate = startTime;
            newRange->endDate = endTime;
            newRange->caseNbr = (char *)malloc(sizeof(char) *
                (strlen(scaseNbr) + 2));
            strcpy_s(newRange->caseNbr, strlen(scaseNbr) + 1, scaseNbr);

```

```

        head->insertSort(newRange, &dateRangeLessThan);
        return;
    }

    ////////////
    // the new start time is before the last record's end time

    // extends the last date range
    // Case 3: (10, 20) (30, 60)
    if (endTime > wtail->endDate) {
        wtail->endDate = endTime;
        return;
    }

    // new range completely enclosed in the last date range
    // Case 4: (10, 20) (30, 80)
    return;
}

    ////////////
    // the new date range starts before wptr's date range

    // the new date range is completely before wptr
    // Case 5: (10, 20) (100, 150)
    if (endTime < wptr->startDate) {

        // The new range starts after tail starts and ends before tail ends
        if (wtail != NULL && wtail->endDate > endTime) {
            return; // nothing to do, it's completely enclosed in tail's date range
        }

        // The new range starts after tail starts but before it ends and it
        // ends after tail ends
        if (wtail != NULL && wtail->endDate > startTime) {
            wtail->endDate = endTime;
            return;
        }

        // The new range starts after tail ends
        newRange = (Parameters::dateRange_t *)
            malloc(sizeof(Parameters::dateRange_t) * 1);

```

```

    newRange->startDate = startTime;
    newRange->endDate = endTime;
    newRange->caseNbr = (char *)malloc(sizeof(char) * (strlen(scaseNbr) + 2));
    strcpy_s(newRange->caseNbr, strlen(scaseNbr) + 1, scaseNbr);
    head->insertSort(newRange, &dateRangeLessThan);
    return;
}

// the new date range starts before wptr's date range and ends after
// wptr's date range starts but before it ends
// Case 6: (10, 20) (60, 80)
if (endTime < wptr->endDate) {
    wptr->startDate = startTime; // extend wptr's date range to beginning of new on
    return;
}

// the date range starts before wptr's date range and ends after it
// i.e., wptr is completely enclosed in the new range -- question - any other rang
wtail = wptr; // we're going to replace wptr's times
liter = head->revIterate(liter);
while (liter != NULL) {
    wptr = head->lvalue(liter);

    if (strcmp(wptr->caseNbr, scaseNbr) != 0) {
        liter = head->revIterate(liter);
        continue;
    }

    // this one is also completely enclosed
    if (wptr->endDate < endTime) {
        liter = head->revIterate(liter);
        head->lerase(wptr);
        free(wptr);
    }
    else break;
}

// everything after wptr is enclosed in the new date range and has been deleted
// Case 7a: (10, 20), (60, 61)
// Case 7b: (10, 20), (60, 61), (63, 65) -- we deleted everything after (60, 61) a
if (liter == NULL) {
    wtail->startDate = startTime;

```

```

        wtail->endDate = endTime;
        return;
    }

    wptr = head->lvalue(liter);

    // there's a gap
    // Case 8a: (10, 20), (60, 61), (80, 90)
    // Case 8b: (10, 20), (60, 61), (63, 65), (80, 90) -- we deleted everything
    //                                     after (60,61) and before (80,90) above
    if (wptr->startDate > endTime) {
        wtail->startDate = startTime;
        wtail->endDate = endTime;
        return;
    }

    // there's no gap
    // Case 9a: (10, 20), (60, 61), (70, 90)
    // Case 9b: (10, 20), (60, 61), (63, 65), (70, 90) -- we deleted everything after
    //                                     (60,61) and before (80,90) above
    wtail->startDate = startTime;
    wtail->endDate = wptr->endDate;
    free(wptr);

    return;
}

```

Appendix H. Miscellaneous Source Code

8.1 TearCPPCPP

Header.

```
#pragma once
#include "sqlite3.h"

/* Standard C++ includes */
#include <stdlib.h>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <tchar.h>
#include <time.h>
#include <math.h>
#include <limits.h>
#include <set>

#ifdef _UNICODE
#define _fputs fputws
#define __fsopen _wfsopen
#define _strcpy_s wcscpy_s
#define _strlen wcslen
#define _sprintf_s swprintf_s
#define _strcat_s wcscat_s
#define _strcmp wcscmp
#else
#define _fputs fputs
#define __fsopen _fsopen
#define _strcpy_s strcpy_s
#define _strlen strlen
#define _sprintf_s sprintf_s
#define _strcat_s strcat_s
#define _strcmp strcmp
#endif

#define SOSLA 1
#define SMTC 2
#define DVSMT 3

#define PLUS 0
#define STAR 1
```



```

#define SINGLE 2

#define NA 0

#define S_REGISTRY 1
#define S_PROCESS 2
#define S_PLASO 3
#define S_CHROME_D 4
#define S_CHROME_V 5
#define S_TSK 6
#define S_EVENT 7
#define S_STRING 8
#define S_LOOP 9
#define S_FIREFOX_D 10
#define S_FIREFOX_V 11

#define GNRC_ACTOR_INSTANCE 9999
#define GNRC_ACTOR 4
#define GNRC_ITEM 5
#define GNRC_ITEMLIST 6

#define R_AT 1
#define R_BEFORE 2
#define R_BETWEEN 3

#define I_FILE 1
#define I_DEVICE 2
#define I_NETWORK 3
#define I_SEARCH 4
#define I_APPLICATION 5
#define I_URL 6
#define I_GENERIC 8
#define I_SERVICE 9
#define I_KEY 10

#define NA 0

#define A_CREATE 1
#define A_ACCESS 2
#define A_MODIFY 3
#define A_CONNECT 4
#define A_FCONNECT 5
#define A_SEARCH 6

```

```
#define A_OPEN 7
#define A_EXECUTE 8
#define A_VISIT 9
#define A_DOWNLOAD 10
#define A_CHANGE 11
#define A_EVENT 12
#define A_REG 13
```

```
#define P_EXPLORER 1
#define P_OFFICE 2
#define P_ADOBERDR 3
#define P_WORDPAD 4
#define P_IE 5
#define P_VCEXPRESS 6
#define P_WORD 7
#define P_EXCEL 8
#define P_POWERPOINT 9
#define P_ACCESS 10
#define P_CHROME 11
#define P_FIREFOX 12
```

```
#define T_DATALOAD 10
```

```
#define T_WIRELESS 2
#define T_WIRED 3
#define T_BROADBAND 4
```

```
typedef struct daterange_type {
    unsigned int startTime;
    unsigned int endTime;
    struct daterange_type *next;
} dateRange;
```

C++.

```
////////////////////////////////////
//                                                                    //
// TEARCPP -- entry point for this console application                //
// Input: [-function] [parameter textfile] [database]                //
```

```

// Output: n/a
// Notes:
// Initial Release: 29-Sep-17
//
// Revision History:
//
////////////////////////////////////
//

```

```

#include "TearCPP.h"
#include "Terms.h"
#include "Strings.h"
#include "Generics.h"
#include "Patterns.h"
#include "AbstractLoops.h"
#include "AbstractStrings.h"
#include "SOSLA.h"
#include "DVSM.T.h"
#include "smtc.h"
#include "Print.h"
#include "BuildDB.h"
#include "ConsolidateStrings.h"
#include "Search.h"
#include "utilities.h"
#include "ItemUtilities.h"
#include "CreateRules.h"

```

```

// First two methods to deal with TCHAR
char * ttoa(TCHAR *t) {
    char *a;
    uint32_t i;

    if (t == NULL) return NULL;
    if (_tcslen(t) == 0) return NULL;

    a = (char *)malloc(sizeof(char) * (_tcslen(t) + 1));
    if (a == NULL) return NULL;

    for (i = 0; i < _tcslen(t); i++) a[i] = (char)t[i];
    a[_tcslen(t)] = 0;

    return a;
}

```

```

void _tstrtoupperIP(TCHAR *lower) {
    unsigned int i;
    if (lower == NULL) return;
    for (i = 0; i < _tcslen(lower); i++)    lower[i] = _totupper(lower[i]);
}

// Main Function
int _tmain(int argc, TCHAR *argv[]) {
    char *parmFilename;
    char *dbName;
    Parameters *parameters;
    Terms *terms;
    Generics *generics;
    Patterns *patterns;
    Strings *strings;
    Utilities *utilities;
    ItemUtilities *itemUtilities;

    parmFilename = ttoa(argv[2]);
    dbName = ttoa(argv[argc - 1]);
    _tstrtoupperIP(argv[1]);

    // Check validity
    if ((_tcscmp(argv[1], L"-STRINGS") != 0) &&
        (_tcscmp(argv[1], L"-EVENTS") != 0) &&
        (_tcscmp(argv[1], L"-SEARCH") != 0) &&
        (_tcscmp(argv[1], L"-CONSOLIDATE") != 0) &&
        (_tcscmp(argv[1], L"-ABSTRACT") != 0) &&
        (_tcscmp(argv[1], L"-LOOPS") != 0) &&
        (_tcscmp(argv[1], L"-SOSLA") != 0) &&
        (_tcscmp(argv[1], L"-SMTC") != 0) &&
        (_tcscmp(argv[1], L"-DVSMT") != 0) &&
        (_tcscmp(argv[1], L"-FINDRULES") != 0) &&
        (_tcscmp(argv[1], L"-PATTERNS") != 0) &&
        (_tcscmp(argv[1], L"-BUILD") != 0) &&
        (_tcscmp(argv[1], L"-EXTRACT") != 0) &&
        (_tcscmp(argv[1], L"-CREATEITEMPATTERNS") != 0) &&
        (_tcscmp(argv[1], L"-PRTRULES") != 0)) {
        _tprintf(TEXT("\n\n\n\n%s [ -patterns | -build | -smtc | -dvsmt | -sosla | -abs
    return 0;
}

```

```

// Give us access to utilities
utilities = new Utilities();

// Create parameters and generics
parameters = new Parameters(utilities);
if (_tcscmp(argv[1], L"-BUILD") == 0) {
    BuildDB *action = new BuildDB();
    action->create(dbName,parameters);
}
parameters->loadParameters(parmFilename,dbName);
generics = new Generics(parameters);
generics->loadGenerics();

itemUtilities = new ItemUtilities(parameters, generics);

patterns = new Patterns(utilities, itemUtilities, parameters, generics);

patterns->loadPatterns();
if (_tcscmp(argv[1], L"-CREATEITEMPATTERNS") == 0) {
    // once you load patterns, you're done
    return 0;
}

terms = new Terms(utilities, parameters, patterns, generics);
terms->loadTerms();
strings = new Strings(utilities, itemUtilities, parameters, terms, generics);
strings->loadStrings();

if (_tcscmp(argv[1], L"-PRTRULES") == 0) {
    Print *action = new Print(utilities, itemUtilities, parameters,
                             patterns, generics, terms, strings);
    action->prtStrings();
    ProductionRules *rules = new ProductionRules(utilities, itemUtilities,
                                                  parameters, patterns, generics, terms, strings);
    rules->loadRules();
    rules->PrintRules();
} else if (_tcscmp(argv[1], L"-PATTERNS") == 0) {
    patterns->createPatterns();
    patterns->writePatterns();
}
else if (_tcscmp(argv[1], L"-BUILD") == 0) {
    BuildDB *action = new BuildDB(parameters,utilities);
    action->load();
}

```

```

}
else if (_tcscmp(argv[1], L"-CONSOLIDATE") == 0) {
    ConsolidateStrings *action = new ConsolidateStrings(utilities, itemUtilities,
        parameters, patterns, generics, terms, strings);
    action->Initialize();
    action->Process();
}
else if (_tcscmp(argv[1], L"-FINDRULES") == 0) {
    ProductionRules *rules = new ProductionRules(utilities, itemUtilities,
        parameters, patterns, generics, terms, strings);
    if (!rules->loadRules()) {
        printf("**ERROR** Did not find ruleset rule! -- exiting\n");
        return 0;
    }
    rules->FindRules();
}
else if (_tcscmp(argv[1], L"-EXTRACT") == 0) {
    ProductionRules *rules = new ProductionRules(utilities, itemUtilities,
        parameters, patterns, generics, terms, strings);
    if (!rules->loadRules()) {
        printf("**ERROR** Did not find ruleset rule! -- exiting\n");
        return 0;
    }
    CreateRules *action = new CreateRules(parameters, patterns, generics,
        terms, strings, rules);
    action->Process();
}
else if (_tcscmp(argv[1], L"-STRINGS") == 0) {
    Print *action = new Print(utilities, itemUtilities, parameters, patterns,
        generics, terms, strings);
    action->prtStrings();
}
else if (_tcscmp(argv[1], L"-EVENTS") == 0) {
    Print *action = new Print(utilities, itemUtilities, parameters, patterns,
        generics, terms, strings);
    action->prtStrings();
    action->prtTimelines();
}
else if (_tcscmp(argv[1], L"-SEARCH") == 0) {
    Search *action = new Search(utilities, itemUtilities, parameters, patterns,
        generics, terms, strings);
    action->searchTimeline();
}
}

```

```

else if (_tcscmp(argv[1], L"-ABSTRACT") == 0) {
    printf("Abstracting data\n");
    AbstractStrings *action = new AbstractStrings(utilities, parameters, patterns,
                                                    strings, generics, terms);
    action->process();
    printf("Abstraction Complete\n");
}
else if (_tcscmp(argv[1], L"-LOOPS") == 0) {
    printf("creating loops\n");
    AbstractLoops *action = new AbstractLoops(utilities, parameters, strings,
                                                generics, terms);
    action->process();
    printf("Abstraction Complete\n");
}
else if (_tcscmp(argv[1], L"-SMTC") == 0) {
    printf("Generating strings...\n");
    smtc *action = new smtc(utilities, parameters, patterns, strings, generics, terms);
    action->process();
    printf("String generation complete\n");
}
else if (_tcscmp(argv[1], L"-DVSMT") == 0) {
    printf("Generating strings...\n");
    dvsmt *action = new dvsmt(utilities, itemUtilities, parameters, patterns, strings,
                                generics, terms);
    action->process();
    printf("String generation complete\n");
}
else if (_tcscmp(argv[1], L"-SOSLA") == 0) {
    printf("Generating strings...\n");
    Sosla *action = new Sosla(utilities, itemUtilities, parameters, patterns, strings,
                                generics, terms);
    action->process();
    printf("String generation complete\n");
}
else {
    return 0;
}

return 0;
}

```

8.2 Parameters

Header.

```
#pragma once
#include <stdlib.h>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <tchar.h>
#include <time.h>
#include <math.h>
#include <limits.h>
#include <stdint.h>
#include "lists.h"
#include "utilities.h"
#include "sqlite3.h"
#include <unordered_map>

#define htExcept_i std::unordered_map<uint32_t, bool>::const_iterator
class Edge;
class VertexContainer;

class Parameters {
public:

    Utilities *utilities;

    typedef struct dateRange_type {
        char *caseNbr;
        uint64_t startDate;
        uint64_t endDate;
    } dateRange_t;

    typedef struct ruleSetEntry_type {
        uint32_t ruleSet;
        char *name;
    } ruleSetEntry_t;

    typedef struct stringExclusion_type {
        char *word;
        stringExclusion_type *next;
    } stringExclusion_t;
```



```

listT<dateRange_t> *dateRange;
listT<ruleSetEntry_t> *ruleSets;
std::unordered_map<uint32_t, bool> *exclusions;
stringExclusion_t *patternExclusions;


uint8_t debugParms;
char *parmFilename;
char *path;
FILE *outFile;
FILE *logFile;
sqlite3 *db;
sqlite3 *productionRules;
listT<VertexContainer> *masterVCList;
listT<Edge> *masterEdgeList;


int maxterm;
uint8_t useItemPatternLists;
uint8_t isDaylightSavingsTime;
uint32_t maxGeneric;
uint32_t maxItemid;
uint32_t nextVertex;
uint32_t nextEdge;
uint32_t existingLevel;
uint64_t startDate; // leading 0 turns this into octal
uint64_t endDate;
char *scaseList;
uint32_t partialMatch;
uint32_t bufferTime;
double epsilon;
uint32_t thresholdSMTC;
double thresholdSimilar;
double thresholdVariation;
double thresholdMain;
uint32_t doMulti;
uint32_t singleString;
uint32_t maxString;
uint32_t timespan;
uint32_t maxItems;
uint32_t showItems;
uint32_t showDetails;
uint32_t showCounts;
uint32_t showZero;

```

```

    uint32_t showTerms;
    uint32_t showStrings;
    uint32_t showValues;
    uint32_t increment;
    uint32_t offsetGMT;
    uint32_t genString;
    char *stringName;
    //  uint32_t ruleSet;

    // status variables
    uint32_t termCount;
    uint32_t abagCount;

public:
    Parameters(Utilities *wutilities) {
        utilities = wutilities;
        masterVCList = new listT<VertexContainer>;
        masterEdgeList = new listT<Edge>;
    }

    ~Parameters();

    void loadParameters(char *parmFilename, char *dbName);
    bool isKeyword(char *winput);
    bool excludeInt(uint32_t winput) {
        return !(exclusions->find(winput) == exclusions->end());
    }
    void indent(uint8_t windent) {
        for (int i = 0; i < windent; i++)
            fprintf(logFile, "\t");
    }
    bool excludePattern(char * winput);

};

```

C++.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: Parameters                                                                                   //
// Input: n/a                                                                                       //
// Output: the parameters parsed into the Parameter class                                         //
// Notes:                                                                                           //
// Initial Release: 29-Sep-17                                                                       //
//                                                                                               //

```

```

// Revision History:
//
//
////////////////////////////////////////////////////////////////

// Will have DB named within the parameter file

#include "Parameters.h"

Parameters::~Parameters() {}

void Parameters::loadParameters(char *wparmFilename, char *dbName) {
    // Purpose: Load the parameters from the passed in flat text parameter file

    FILE *parmFile;
    char *outFilename, *logFilename, *productionDBName;
    char *winput, exec_string[2000];
    uint32_t wstarttime, wendtime;
    dateRange_t *wdateRange;
    ruleSetEntry_t *wruleSet;
    int rc;
    bool isExclusions;
    bool isPatternExclusions;
    uint32_t excludeInt;
    sqlite3_stmt *stmt_selItem;
    stringExclusion_t *wexclusion, *prevExclusion;

    //////////////////////////////////////////////////////////////////

    // Open parameter file
    if (wparmFilename == NULL) return;
    parmFilename = (char *)malloc(sizeof(char) * (strlen(wparmFilename) + 2));
    strcpy_s(parmFilename, strlen(wparmFilename) + 1, wparmFilename);
    fopen_s(&parmFile, parmFilename, "r");
    if (parmFile == NULL) {
        printf("Could not open parameter file %s\n", parmFilename);
        return;
    }

    // Open the database
    rc = sqlite3_open_v2(dbName, &db, SQLITE_OPEN_READWRITE, NULL);
    if (rc != SQLITE_OK) { printf("Error: %s\n", sqlite3_errmsg(db)); return; }

    // Create parameter record

```

```

// Set the status variables
termCount = 0;
abagCount = 0;
patternExclusions = NULL;

// Get the maximum itemid in the DB as a starting point for new itemids
sprintf_s(exec_string, 2000, "SELECT max(itemid) FROM Items;");
rc = sqlite3_prepare_v2(db, exec_string, 2000, &stmt_selItem, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Select Item Statement\n", rc);
    return;
}
sqlite3_reset(stmt_selItem);
rc = sqlite3_step(stmt_selItem);
maxItemId = (rc == SQLITE_ROW ? sqlite3_column_int(stmt_selItem, 0) + 1 : 1);
sqlite3_finalize(stmt_selItem);

// default values before checking parameter file
stringName = NULL;
genString = false;
isExclusions = false;
isPatternExclusions = false;
isDaylightSavingsTime = 1;
useItemPatternLists = true;
debugParms = 0; // level of debugging (-1 = time test)
outFile = stdout;
logFile = stderr;
epsilon = 3.0;
thresholdSMTC = 10000;
thresholdSimilar = 0.30;
doMulti = 0;
singleString = 0;
maxString = 40;
productionDBName = NULL;
existingLevel = 10;
startDate = utilities->convertToUnixTime(1012001); // leading 0 turns this to octa
endDate = utilities->convertToUnixTime(12312099);
scaseList = NULL;
partialMatch = 0;
bufferTime = 30;
maxItems = 0;
nextVertex = 1;

```

```

nextEdge = 1;
showDetails = 1;
showItems = 1;
timespan = 300;
showCounts = 1;
showValues = 0;
showZero = 1;
showTerms = 1;
showStrings = 1;
increment = 10000;
offsetGMT = 0;
path = NULL;
dateRange = new listT<dateRange_t>;
ruleSets = new listT<ruleSetEntry_t>;
exclusions = new std::unordered_map<uint32_t, bool>;
patternExclusions = NULL;
prevExclusion = NULL;
// ruleSet = 0;

wstarttime = wendtime = 0;

// Load the parameters from the parameter file
while (!feof(parmFile)) {
    winput = utilities->get_cfield(parmFile);

    // skip comments
    if ((strlen(winput) > 2) && (winput[0] == '/') && (winput[1] == '/')) {
        free(winput); winput = NULL;
        continue;
    }

    if (isKeyword(winput) && utilities->strcmpU("Exclusions", winput)) isExclusions
    if (isKeyword(winput) && utilities->strcmpU("ExcludePatterns", winput)) isPatte

    if (!utilities->strcmpU("Debug", winput))
        debugParms = (uint8_t)utilities->get_ifield(parmFile);
    else if (!utilities->strcmpU("ExistingLevel", winput))
        existingLevel = utilities->get_ifield(parmFile);
    else if (!utilities->strcmpU("Increment", winput))
        increment = utilities->get_ifield(parmFile);
    else if (!utilities->strcmpU("casenbr", winput))
        scaseList = utilities->get_cfield(parmFile);
    else if (!utilities->strcmpU("Path", winput))

```

```

    path = utilities->get_cfield(parmFile);
else if (!utilities->strcmpU("GMTOffset", winput))
    offsetGMT = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("StartDate", winput))
    startDate = utilities->convertToUnixTime(utilities->get_ifield(parmFile));
else if (!utilities->strcmpU("EndDate", winput))
    endDate = utilities->convertToUnixTime(utilities->get_ifield(parmFile));
else if (!utilities->strcmpU("StartTime", winput))
    wstarttime = utilities->convertToUnixSeconds(utilities->get_ifield(parmFile));
else if (!utilities->strcmpU("EndTime", winput))
    wendtime = utilities->convertToUnixSeconds(utilities->get_ifield(parmFile));
else if (!utilities->strcmpU("UseItemPatternLists", winput))
    useItemPatternLists = (utilities->get_ifield(parmFile) == 1 ? true : false);

// Grammar Build Parameters
else if (!utilities->strcmpU("PartialMatch", winput))
    partialMatch = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("BufferTime", winput))
    bufferTime = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("MultiItem", winput))
    doMulti = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("Epsilon", winput))
    epsilon = utilities->get_ffield(parmFile);
else if (!utilities->strcmpU("ThresholdScore", winput))
    thresholdSMTC = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("ThresholdMain", winput))
    thresholdMain = utilities->get_ffield(parmFile);
else if (!utilities->strcmpU("ThresholdVariation", winput))
    thresholdVariation = utilities->get_ffield(parmFile);
else if (!utilities->strcmpU("ThresholdSimilar", winput))
    thresholdSimilar = utilities->get_ffield(parmFile);
else if (!utilities->strcmpU("SingleString", winput))
    singleString = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("MaximumStringLength", winput))
    maxString = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("ShowValues", winput))
    showValues = utilities->get_ifield(parmFile);

// Generate Strings
else if (!utilities->strcmpU("GenerateString", winput))
    genString = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("StringName", winput))
    stringName = utilities->get_cfield(parmFile);

```

```

// Print Parameters
else if (!utilities->strcmpU("outFilename", wininput)) {
    outFilename = utilities->get_cfield(parmFile);
    fopen_s(&(outFile), outFilename, "w");
    if (outFile == NULL) {
        printf("Could not open outfile %s\n", outFilename);
        return;
    }
}
else if (!utilities->strcmpU("logFilename", wininput)) {
    logFilename = utilities->get_cfield(parmFile);
    fopen_s(&(logFile), logFilename, "w");
    if (logFile == NULL) {
        printf("Could not open logfile %s\n", logFilename);
        return;
    }
} else if (!utilities->strcmpU("timespan", wininput))
    timespan = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("maxItem", wininput))
    maxItems = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("details", wininput))
    showDetails = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("items", wininput))
    showItems = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("Counts", wininput))
    showCounts = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("Zeros", wininput))
    showZero = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("Terms", wininput))
    showTerms = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("Strings", wininput))
    showStrings = utilities->get_ifield(parmFile);
else if (!utilities->strcmpU("DateRange", wininput)) {
    wdateRange = (dateRange_t *)malloc(sizeof(dateRange) * 1);
    wdateRange->caseNbr = utilities->get_cfield(parmFile);
    wdateRange->startDate = utilities->convertToUnixTime(utilities->get_ifield(parmFile));
    wdateRange->startDate += utilities->convertToUnixSeconds(utilities->get_ifield(parmFile));
    wdateRange->endDate = utilities->convertToUnixTime(utilities->get_ifield(parmFile));
    wdateRange->endDate += utilities->convertToUnixSeconds(utilities->get_ifield(parmFile));
    dateRange->insertRear(wdateRange);
}
}

```

```

// Production Rules
else if (!utilities->strcmpU("Ruleset", winput)) {
    wruleSet = (ruleSetEntry_t *)malloc(sizeof(ruleSetEntry_t) * 1);
    wruleSet->ruleSet = utilities->get_ifield(parmFile);
    wruleSet->name = utilities->get_cfield(parmFile);
    ruleSets->insertRear(wruleSet);
}
// Production Rules DB
else if (!utilities->strcmpU("productionRules", winput)) {
    productionDBName = utilities->get_cfield(parmFile);
    rc = sqlite3_open_v2(productionDBName, &productionRules, SQLITE_OPEN_READWRITE, NULL);
    if (rc != SQLITE_OK) {
        printf("Error: could not open Production Rules: %s\n",
            sqlite3_errmsg(productionRules));
        return;
    }
}

// Exclusions
else if (!utilities->strcmpU("Exclusions", winput)) isExclusions = true;
else if (isExclusions) {
    excludeInt = atoi(winput);
    (*exclusions)[excludeInt] = true;
}
else if (!utilities->strcmpU("ExcludePatterns", winput)) isPatternExclusions = true;
else if (isPatternExclusions) {
    wexclusion = (stringExclusion_t *)malloc(sizeof(stringExclusion_t) * 1);
    wexclusion->next = NULL;
    wexclusion->word = winput;
    if (prevExclusion != NULL) prevExclusion->next = wexclusion;
    if (patternExclusions == NULL) patternExclusions = wexclusion;
    prevExclusion = wexclusion;
    continue;
}

free(winput); winput = NULL;
}

// Convert and store the date ranges
for (void * dateRangeIter = dateRange->getFront(); dateRangeIter != NULL;
    dateRangeIter = dateRange->revIterate(dateRangeIter)) {
    char stime[80], etime[80];
    time_t wtimer;

```



```

    struct tm timeinfo;
    dateRange_t *wdateRange;

    wdateRange = dateRange->lvalue(dateRangeIter);
    wtimer = (time_t)wdateRange->startDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(stime, 80, "%x %X", &timeinfo);

    wtimer = (time_t)wdateRange->endDate;
    localtime_s(&timeinfo, &wtimer);
    strftime(etime, 80, "%x %X", &timeinfo);
    if (debugParms >= 4) fprintf(logFile, "date range = %s - %s\n", stime, etime)
}

if (debugParms >= 4) fprintf(logFile, "maxItemid = %u\n", maxItemid);
fclose(parmFile);

// Format the path correctly
if (path == NULL) {
    path = (char *)malloc(sizeof(char) * 4);
    strcpy_s(path, 3, ".\\");
} else
    utilities->fixpath(path);

// Add the time into the start and end date
startDate += wstarttime;
endDate += wendtime;

}

////////////////////////////////////
// Return true if passed a valid keyword for the parameter file //
////////////////////////////////////
bool Parameters::isKeyword(char *winput) {
    if (!utilities->strcmpU("Debug", winput) ||
        !utilities->strcmpU("ExistingLevel", winput) ||
        !utilities->strcmpU("Increment", winput) ||
        !utilities->strcmpU("casenbr", winput) ||
        !utilities->strcmpU("StartDate", winput) ||
        !utilities->strcmpU("EndDate", winput) ||
        !utilities->strcmpU("StartTime", winput) ||

```

```

!utilities->strcmpU("EndTime", winput) ||
!utilities->strcmpU("ThresholdSimilar", winput) ||
!utilities->strcmpU("PartialMatch", winput) ||
!utilities->strcmpU("BufferTime", winput) ||
!utilities->strcmpU("MultiItem", winput) ||
!utilities->strcmpU("Epsilon", winput) ||
!utilities->strcmpU("ThresholdScore", winput) ||
!utilities->strcmpU("ThresholdMain", winput) ||
!utilities->strcmpU("ThresholdVariation", winput) ||
!utilities->strcmpU("ThresholdSimilar", winput) ||
!utilities->strcmpU("SingleString", winput) ||
!utilities->strcmpU("MaximumStringLength", winput) ||
!utilities->strcmpU("ShowValues", winput) ||
!utilities->strcmpU("outFilename", winput) ||
!utilities->strcmpU("logFilename", winput) ||
!utilities->strcmpU("timespan", winput) ||
!utilities->strcmpU("maxItem", winput) ||
!utilities->strcmpU("details", winput) ||
!utilities->strcmpU("items", winput) ||
!utilities->strcmpU("Counts", winput) ||
!utilities->strcmpU("Zeros", winput) ||
!utilities->strcmpU("Terms", winput) ||
!utilities->strcmpU("Strings", winput) ||
!utilities->strcmpU("GenerateString", winput) ||
!utilities->strcmpU("StringName", winput) ||
!utilities->strcmpU("Exclusions", winput) ||
!utilities->strcmpU("ExcludePatterns", winput) ||
!utilities->strcmpU("Ruleset", winput) ||
!utilities->strcmpU("Instances", winput) ||
!utilities->strcmpU("Searches", winput) ||
!utilities->strcmpU("Replacements", winput))
return true;
return false;
}

bool Parameters::excludePattern(char *winput) {
    stringExclusion_t *wptr;
    for (wptr = patternExclusions; wptr != NULL; wptr = wptr->next) {
        if (strlen(wptr->word) > 1 && utilities->matchu(wptr->word, winput)) {
            return true;
        }
    }
    return false;
}

```

8.3 BuildDB

Header.

```
#pragma once
#include <stdlib.h>
#include <windows.h>
#include "TearCPP.h"
#include "Parameters.h"
#include "Utilities.h"
#include "glib.h"

class BuildDB
{
private:
    WIN32_FIND_DATA *fd;

    typedef struct registryType {
        char *registryKey;
        char *keyType;
        char *charValue;
        uint32_t timeStamp;
    } registry_t;

    typedef struct searchType {
        uint32_t itemid;
        char *searchKey;
        uint8_t application;
        uint8_t traceType;
        uint32_t traceid;
    } search_t;

    typedef struct deviceType {
        uint32_t itemid;
        char *deviceid;
        char *serialNumber;
        char *friendlyName;
        char *guid;
        char *mac;
        uint8_t traceType;
        uint32_t traceid;
    } device_t;
```

```

typedef struct itemType {
    uint32_t itemid;
    uint8_t itemtypeid;
    uint8_t traceType;
    uint32_t traceid;
    char *sort;
} item_t;

typedef struct serviceType {
    uint32_t itemid;
    char *serviceid;
    uint8_t traceType;
    uint32_t traceid;
} service_t;

typedef struct fileType {
    uint32_t itemid;
    uint32_t hash;
    char *fileName;
    uint8_t traceType;
    uint32_t traceid;
} file_t;

typedef struct applicationType {
    uint32_t itemid;
    char *applicationName;
    uint32_t count;
    uint8_t traceType;
    uint32_t traceid;
} application_t;

typedef struct urlType {
    uint32_t itemid;
    char *urlName;
    uint8_t traceType;
    uint32_t traceid;
} url_t;

typedef struct keyType {
    uint32_t itemid;
    char *keyName;
    uint8_t traceType;
    uint32_t traceid;
}

```

```

} key_t;

typedef struct addht_type {
    FILE *wfile;
} addht_t;

Parameters *parameters;
Utilities *utilities;

sqlite3_stmt *stmt_selActor;
sqlite3_stmt *stmt_insActor;
sqlite3_stmt *stmt_selActor2;
sqlite3_stmt *stmt_insActor2;
sqlite3_stmt *stmt_insActions;
sqlite3_stmt *stmt_selAppexec;
sqlite3_stmt *stmt_insAppexec;
sqlite3_stmt *stmt_insE;
sqlite3_stmt *stmt_selE;

sqlite3 *events;
sqlite3_stmt *stmt_selEM;
sqlite3_stmt *stmt_selEM2;
sqlite3_stmt *stmt_selEMs;
sqlite3_stmt *stmt_selEMs2;
sqlite3_stmt *stmt_updEM;
sqlite3_stmt *stmt_updEM2;
sqlite3_stmt *stmt_resetEM;
sqlite3_stmt *stmt_resetEM2;
sqlite3_stmt *stmt_selRegistry;
sqlite3_stmt *stmt_insRegistryText;
sqlite3_stmt *stmt_insRegistryChar;
sqlite3_stmt *stmt_selDevice;
sqlite3_stmt *stmt_selDevice2;
sqlite3_stmt *stmt_updDevice;
sqlite3_stmt *stmt_insDevice;
sqlite3_stmt *stmt_insDevice2;
sqlite3_stmt *stmt_selNetwork;
sqlite3_stmt *stmt_insNetwork;

FILE *eventFile, *timeline, *eventSupport, *activities, *mruList, *registry;
uint32_t nextEvent, nextTimeline, nextActivity, nextItem, nextRegistry;

GHashTable *htItems, *htFiles, *htUrls, *htApplications, *htSearches, *htKeys, *ht

```

```

uint32_t insert_item(uint8_t witemtypeid, uint8_t tracetypeid,
                    uint32_t traceid, char *wsort);
uint32_t insert_event_support(uint32_t wactivityID, uint8_t winfo_type,
                             char * winfo, uint8_t wtracetypeid, uint32_t wtraceid);
uint32_t insert_activity(uint32_t wactorid, uint32_t wapplicationid,
                        uint16_t wactionid, uint32_t witemid, uint8_t wsourceid,
                        uint8_t tracetypeid, uint32_t traceid);
uint32_t insert_event(uint32_t woid, uint32_t n32time, uint32_t wappexec,
                    uint32_t wactor, uint32_t waction, char *wmessage);
uint32_t insert_timeline(uint8_t wtimelinetypeid, uint32_t wactivity1id,
                        uint32_t wactivity2id, uint8_t wrelationshipid, uint32_t wtime1,
                        uint32_t wtime2, uint8_t wsourceid, uint8_t wtracetypeid,
                        uint32_t wtraceid, TCHAR *path);
uint32_t insert_mru(uint32_t wactivityid, uint32_t wseqid, uint32_t wtime,
                  uint8_t wsourceid, uint8_t wtracetypeid, uint32_t wtraceid);
uint32_t insert_registry(char * wregistryKey, char *wkeyType, char *wcharValue, ui

uint32_t get_appexec(char * wapplication, uint8_t wtracetypeid, uint32_t wtraceid)
uint32_t get_application(char * wapplication, uint32_t wcount, uint8_t wtracetypeid)
uint32_t get_file(char * wfilename, uint8_t wtracetypeid, uint32_t wtraceid);
uint32_t get_url(char * wurl, uint8_t wtracetypeid, uint32_t wtraceid);
uint32_t get_key(char * wkey, uint8_t wtracetypeid, uint32_t wtraceid);
uint32_t get_search(char * wsearchkey, uint8_t wapplication, uint8_t wtracetypeid,
uint32_t get_actor(TCHAR *wname);
uint32_t get_actor2(char *wuser);

uint32_t last_insert(sqlite3 *db);

int processDir(TCHAR *path);
void BuildDB::doDBAction(char *wstmt);

public:
    BuildDB();
    BuildDB(Parameters *parameters, Utilities *wutilities);
    ~BuildDB();
    void create(char *dbName, Parameters *parameters);
    void load();
};

```

C++.

```
/////////////////////////////////////////////////////////////////
//                                                                    //
// Class: BuildDB                                                    //
// Input: n/a                                                        //
// Output: a database with the low level events and support records //
// Notes:                                                            //
//     1. There are two primary methods - one to create the database //
//         and one to load the data into it.                        //
// Initial Release: 29-Sep-17                                         //
//                                                                    //
// Revision History:                                                //
//                                                                    //
/////////////////////////////////////////////////////////////////
#include "BuildDB.h"

BuildDB::BuildDB() {}

BuildDB::BuildDB(Parameters *wparameters, Utilities *wutilities) {

    // Used to create the SQL statements
    char exec_string[2000];
    int rc;

    ///////////////////////////////////////////////////////////////////

    // Initialize Variables
    fd = (WIN32_FIND_DATA *)malloc(sizeof(WIN32_FIND_DATA));
    parameters = wparameters;
    utilities = wutilities;

    // Set the next available ids
    nextEvent = nextTimeline = 1;
    nextActivity = 1;
    nextRegistry = 1;
    nextItem = 41;

    // Open the dump files; loading these records real time slows this to a crawl.
    // It's much faster to use dump files and then use .import commands from the
    // SQL command line
    fopen_s(&eventFile, "../events.out", "w");
    fopen_s(&activities, "../activities.out", "w");
    fopen_s(&timeline, "../timeline.out", "w");
}
```

```

fopen_s(&eventSupport, "../eventSupport.out", "w");
fopen_s(&mruList, "../mruList.out", "w");
fopen_s(&registry, "../registry.out", "w");

// Create the hash tables that hold the information until they're dumped
htItems = g_hash_table_new(g_int_hash, g_int_equal);
htFiles = g_hash_table_new(g_str_hash, g_str_equal);
htApplications = g_hash_table_new(g_str_hash, g_str_equal);
htUrls = g_hash_table_new(g_str_hash, g_str_equal);
htKeys = g_hash_table_new(g_str_hash, g_str_equal);
htSearches = g_hash_table_new(g_str_hash, g_str_equal);

// Create the SQL statements

// Additional actors come from the users who execute the low level events
sprintf_s(exec_string, 1000, "SELECT ActorID FROM actors WHERE Description = ?1;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selActor2, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "INSERT INTO actors (Description) VALUES (?1);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insActor2, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare it\n", rc);
    return;
}

// Additional actions come from the event log messages
sprintf_s(exec_string, 1000, "INSERT INTO actions (Description) VALUES (?1);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insActions, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare it\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "SELECT ActorID FROM actors WHERE Description = ?1;");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selActor, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

```



```

}

sprintf_s(exec_string, 1000, "INSERT INTO actors (Description) VALUES (?1);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insActor, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare it\n", rc);
    return;
}

// Application that performed the action
sprintf_s(exec_string, 1000, "SELECT ApplicationID FROM appexecs WHERE Description = ?1");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selAppexec, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select appexecs statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "INSERT INTO appexecs (Description, TraceType, TraceID) VALUES (?1, ?2, ?3);");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insAppexec, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Applications statement\n", rc);
    return;
}

// Registry
sprintf_s(exec_string, 1000, "SELECT RegistryKey FROM registry WHERE RegistryKey = ?1");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selRegistry, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

// Devices
sprintf_s(exec_string, 1000, "SELECT FriendlyName, GUID, MAC, ItemID FROM devices WHERE ItemID = ?1");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selDevice, 0);
if (rc != SQLITE_OK) {
    printf("xError: %i. couldn't prepare select device statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "SELECT ItemID FROM devices WHERE GUID = ?1 AND MAC = ?2");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selDevice2, 0);

```

```

if (rc != SQLITE_OK) {
    printf("xError: %i. couldn't prepare select device statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "UPDATE devices SET FriendlyName = ?1, GUID = ?2, MAC
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_updDevice, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Update devices statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "INSERT INTO devices (ItemID, DeviceID, SerialNumber,
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insDevice, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Devices statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "INSERT INTO devices (ItemID, DeviceID, SerialNumber,
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insDevice2, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Devices statement\n", rc);
    return;
}

// Networks
sprintf_s(exec_string, 1000, "SELECT ItemID FROM networks WHERE NetworkID = ?1 AND
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selNetwork, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select networks statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "INSERT INTO networks (ItemID, NetworkID, Type, Manag
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_insNetwork, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare Insert Networks statement\n", rc);
    return;
}

// Event Messages

```

```

sprintf_s(exec_string, 1000, "SELECT EventID, Action FROM events WHERE EventMessage = ?1 AND EventID = ?2");
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt_selE, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

// Open the event message database
_tprintf(L"Opening up event message database\n");
rc = sqlite3_open_v2(".\\eventmessages.sql", &events, SQLITE_OPEN_READWRITE, NULL);
if (rc != SQLITE_OK) {
    printf("Error: %i -- %s\n", rc, sqlite3_errmsg(events));
    return;
}

// SQL statements for the event message database
sprintf_s(exec_string, 1000, "SELECT message, action FROM eventmessages WHERE id = ?1 AND EventID = ?2");
rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_selEM, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "SELECT message, action FROM eventmessages WHERE id2 = ?1 AND EventID = ?2");
rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_selEM2, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "SELECT count(*) FROM eventmessages WHERE id = ?1 AND EventID = ?2");
rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_selEMs, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

sprintf_s(exec_string, 1000, "SELECT count(*) FROM eventmessages WHERE id2 = ?1 AND EventID = ?2");
rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_selEMs2, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select statement\n", rc);
    return;
}

```

```

    sprintf_s(exec_string, 1000, "UPDATE eventmessages SET action = ?3 WHERE id = ?1 A
    rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_updEM, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare select statement\n", rc);
        return;
    }

    sprintf_s(exec_string, 1000, "UPDATE eventmessages SET action = ?3 WHERE id2 = ?1
    rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_updEM2, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare select statement\n", rc);
        return;
    }

    sprintf_s(exec_string, 1000, "UPDATE eventmessages SET action = %i;", A_EVENT);
    rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_resetEM, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare select statement\n", rc);
        return;
    }

    sprintf_s(exec_string, 1000, "UPDATE eventmessages SET keyname = upper(keyname);")
    rc = sqlite3_prepare_v2(events, exec_string, 1000, &stmt_resetEM2, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare select statement\n", rc);
        return;
    }
}

BuildDB::~BuildDB() {}

void BuildDB::create(char *dbName, Parameters *wparameters) {
    // Purpose: Create the main database

    int rc;

    //////////////////////////////////////

    // Initialize variables

```

```

parameters = wparameters;

printf("Creating Database %s\n", dbName);

// Open the database
rc = sqlite3_open_v2(dbName, &(parameters->db), SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE, NULL);
if (rc != SQLITE_OK) {
    printf("Error: %s\n", sqlite3_errmsg(parameters->db));
    return;
}

// Load the schema and some standard records

doDBAction("DROP TABLE sources;");
doDBAction("CREATE TABLE sources(SourceID INT1 NOT NULL, Description CHAR(50) NOT NULL);");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(0, 'N/A');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(1, 'Registry');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(2, 'Processes');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(3, 'Plaso');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(4, 'Chrome Downloads');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(5, 'Chrome Visits');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(6, 'The Sleuth Kit');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(7, 'Event Logs');");
doDBAction("INSERT INTO sources(SOURCEID, Description) VALUES(8, 'Strings');");
doDBAction("INSERT INTO sources(SOURCEID, Description) VALUES(9, 'Loops');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(10, 'Firefox Downloads');");
doDBAction("INSERT INTO sources(SourceID, Description) VALUES(11, 'Firefox Visits');");

// timeline files
doDBAction("DROP TABLE timeline;");
doDBAction("CREATE TABLE timeline(TimelineID INTEGER PRIMARY KEY AUTOINCREMENT, TimelineDescription TEXT);");
doDBAction("Create INDEX timelineTimelineid ON timeline (timelineid);");

doDBAction("DROP TABLE timelinetypes;");
doDBAction("CREATE TABLE timelinetypes(TimelineTypeID INT1 NOT NULL, Description TEXT);");
doDBAction("INSERT INTO timelinetypes(TimelineTypeID, Description) VALUES(10, 'Timeline Types');");

doDBAction("DROP TABLE relationships;");
doDBAction("CREATE TABLE relationships(RelationshipID INT1 NOT NULL, Description TEXT);");
doDBAction("INSERT INTO relationships(RelationshipID, Description) VALUES(1, 'A');");
doDBAction("INSERT INTO relationships(RelationshipID, Description) VALUES(2, 'B');");
doDBAction("INSERT INTO relationships(RelationshipID, Description) VALUES(3, 'C');");

```

```

// activity files (also used in terms)
doDBAction("DROP TABLE activities;");
doDBAction("CREATE TABLE activities(ActivityID INTEGER PRIMARY KEY AUTOINCREMENT,
doDBAction("Create INDEX activitiesActivityId ON activities (activityID);");
doDBAction("Create INDEX activitiesActorAction ON activities (actorid, application
doDBAction("INSERT INTO activities(ActivityID, ActorID, ApplicationID, ActionID, I

doDBAction("DROP TABLE actors;");
doDBAction("CREATE TABLE actors(ActorID INTEGER PRIMARY KEY AUTOINCREMENT, Descrip
doDBAction("INSERT INTO actors(ActorID, Description) VALUES(0, 'N/A');");

doDBAction("DROP TABLE actions;");
doDBAction("CREATE TABLE actions(ActionID INTegeR PRIMARY KEY AUTOINCREMENT, Descr
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(0, 'N/A/');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(1, 'Created');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(2, 'Accessed');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(3, 'Modified');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(4, 'Connected');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(5, 'First Connect
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(6, 'Searched');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(7, 'Opened');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(8, 'Executed');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(9, 'Visit');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(10, 'Download');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(11, 'Changed');");
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(12, 'Event Occurr
doDBAction("INSERT INTO actions(ActionID, Description) VALUES(13, 'Registry Key

doDBAction("DROP TABLE appexecs;");
doDBAction("CREATE TABLE appexecs(ApplicationID INTEGER PRIMARY KEY AUTOINCREMENT,
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID

```

```

doDBAction("INSERT INTO appexecs(ApplicationID, Description, TraceType, TraceID
doDBAction("DROP TABLE itemtypes;");
doDBAction("CREATE TABLE itemtypes(ItemTypeID INT1 NOT NULL, Description CHAR(50)
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(0, 'N/A');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(1, 'Files');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(2, 'Devices');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(3, 'Networks');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(4, 'Searches');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(5, 'Applications');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(6, 'Urls');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(7, 'Applications');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(8, 'Generics');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(9, 'Services');");
doDBAction("INSERT INTO itemtypes(ItemTypeID, Description) VALUES(10, 'Registry');");

doDBAction("DROP TABLE items;");
doDBAction("CREATE TABLE items(ItemID INTEGER PRIMARY KEY AUTOINCREMENT, ItemTypeID
doDBAction("Create INDEX itemItemids ON items (itemid);");
doDBAction("INSERT INTO items(ItemID, ItemTypeID, TraceType, TraceID, sort) VALUES
doDBAction("Create TABLE itemPatterns(ItemID INTEGER, PatternID INT4);");
doDBAction("Create INDEX itemPs ON itemPatterns (itemid,patternID);");

// additional item information
doDBAction("DROP TABLE devices;");
doDBAction("CREATE TABLE devices(ItemID INT4, DeviceID CHAR(250) NOT NULL, SerialN
doDBAction("DROP TABLE services;");
doDBAction("CREATE TABLE services(ItemID INT4, ServiceID CHAR(250) NOT NULL, Trace
doDBAction("DROP TABLE networks;");
doDBAction("CREATE TABLE networks(ItemID INT4, NetworkID CHAR(250) NOT NULL, Type
doDBAction("DROP TABLE searches;");
doDBAction("CREATE TABLE searches(ItemID INT4, SearchKey CHAR(250) NOT NULL, Appli
doDBAction("DROP TABLE files;");
doDBAction("CREATE TABLE files(ItemID INT4, Hash INT4, FileName CHAR(250) NOT NULL
doDBAction("Create INDEX fileFileNames ON files (filename);");
doDBAction("Create INDEX fileItems ON files (itemid);");
doDBAction("DROP TABLE genericItems;");
doDBAction("CREATE TABLE genericItems(genericItem INT4, itemID INT4);");
doDBAction("DROP TABLE applications;");
doDBAction("CREATE TABLE applications(ItemID INT4, ApplicationName CHAR(250) NOT N
doDBAction("DROP TABLE urls;");
doDBAction("CREATE TABLE urls(ItemID INT4, UrlName CHAR(250) NOT NULL, TraceType I
doDBAction("DROP TABLE keys;");

```

```

doDBAction("CREATE TABLE keys(ItemID INT4, KeyName CHAR(250) NOT NULL, TraceType INT4);");
doDBAction("Create INDEX keyKeynames ON files (keyname);");
doDBAction("Create INDEX keyItems ON files (itemid);");
doDBAction("DROP TABLE downloads;");
doDBAction("CREATE TABLE downloads(ActivityID INT4 NOT NULL, SeqId INT2 NOT NULL,

// terms and strings -- terms use support info in activities (e.g., actor, action,
doDBAction("DROP TABLE strings;");
doDBAction("CREATE TABLE Strings(id INTEGER PRIMARY KEY AUTOINCREMENT, stringName CHAR(250) NOT NULL);");
doDBAction("DROP TABLE stringTerms;");
doDBAction("CREATE TABLE StringTerms(id INTEGER, position INT4, term INT4, type INT4);");
doDBAction("DROP TABLE terms;");
doDBAction("CREATE TABLE Terms(id INTEGER PRIMARY KEY AUTOINCREMENT, actor INT4, action INT4, term INT4);");
doDBAction("DROP TABLE wildcards;");
doDBAction("CREATE TABLE wildcards(wildcard INT1, description CHAR(250) NOT NULL);");
doDBAction("INSERT INTO wildcards(wildcard, description) VALUES(0, 'Plus');");
doDBAction("INSERT INTO wildcards(wildcard, description) VALUES(1, 'Star');");
doDBAction("INSERT INTO wildcards(wildcard, description) VALUES(2, 'Single');");
doDBAction("DROP TABLE patterns;");
doDBAction("CREATE TABLE patterns(id INTEGER PRIMARY KEY AUTOINCREMENT, pattern CHAR(250) NOT NULL);");
doDBAction("DROP TABLE patternGenerics;");
doDBAction("CREATE TABLE patternGenerics(id INT4, itemid INT4);");
doDBAction("INSERT INTO patterns(id, pattern) VALUES(1, '*');");

// loops -- occurs after initial timeline is abstracted
doDBAction("DROP TABLE loops;");
doDBAction("CREATE TABLE Loops(loopID INTEGER PRIMARY KEY AUTOINCREMENT, startID INT4);");
doDBAction("Create INDEX loopsLoopID ON loops (loopID);");
doDBAction("DROP TABLE loopSupport;");
doDBAction("CREATE TABLE LoopSupport(loopID INTEGER, timelineID INT4);");
doDBAction("Create INDEX loopSupportLoopID ON loopSupport (loopid, timelineid);");

// string instances -- occurs after initial timeline is abstracted
doDBAction("DROP TABLE stringInstances;");
doDBAction("CREATE TABLE StringInstances(instanceId INTEGER PRIMARY KEY AUTOINCREMENT, stringId INT4);");
doDBAction("Create INDEX StringInstancesInstanceId ON stringInstances (instanceID);");
doDBAction("Create INDEX StringInstancesStringId ON stringInstances (stringId);");
doDBAction("DROP TABLE stringSupport;");
doDBAction("CREATE TABLE StringSupport(instanceId INT4, genericType INT4, GenericId INT4);");
doDBAction("Create INDEX stringSupportInstances ON StringSupport (instanceId, genericType);");
doDBAction("Create INDEX stringSupportGeneric ON StringSupport (instanceId, genericType, GenericId);");
doDBAction("DROP TABLE generics;");
doDBAction("CREATE TABLE Generics(genericType INT4, Description CHAR(255));");

```



```

        doDBAction("INSERT INTO generics(genericType, description) VALUES(5, 'Item');");
        doDBAction("INSERT INTO generics(genericType, description) VALUES(6, 'ItemList'");
doDBAction("DROP TABLE StringXref;");
doDBAction("CREATE TABLE StringXref(instanceId INT4, orderid INT4, timelineId INT4);");
doDBAction("Create INDEX stringXrefInstances ON stringXref (instanceId, orderid);");
doDBAction("Create INDEX stringXrefTI ON stringXref (timelineid, instanceId);");
doDBAction("DROP TABLE stringSupportXref;");
doDBAction("CREATE TABLE StringSupportXref(instanceId INT4, orderId CHAR(255), oldId INT4);");
doDBAction("Create INDEX stringSupportXrefInstances ON stringSupportXref (instanceId, orderId);");
doDBAction("Create INDEX stringSupportXrefIOO ON stringSupportXref (instanceId, orderId);");

// addition event information
doDBAction("DROP TABLE registry;");
doDBAction("CREATE TABLE registry(RegID INTEGER PRIMARY KEY NOT NULL, RegistryKey CHAR(255));");
doDBAction("DROP TABLE processes;");
doDBAction("CREATE TABLE processes(ProcessID INTEGER PRIMARY KEY AUTOINCREMENT, ProcessName CHAR(255));");
doDBAction("DROP TABLE events;");
doDBAction("CREATE TABLE events(EventID INTEGER PRIMARY KEY NOT NULL, OriginalEventID INT4, EventType INT1, Description CHAR(255));");
doDBAction("DROP TABLE mrulist;");
doDBAction("CREATE TABLE mrulist(ActivityID INT4 NOT NULL, Sequence INT2 NOT NULL, Description CHAR(255));");
doDBAction("DROP TABLE support_types;");
doDBAction("CREATE TABLE support_types(InfoTypeID INT1 NOT NULL, Description CHAR(255));");
doDBAction("DROP TABLE event_support;");
doDBAction("CREATE TABLE event_support(ActivityID INT4 NOT NULL, InfoTypeID INT1 NOT NULL, Description CHAR(255));");
doDBAction("DROP TABLE mft;");
doDBAction("CREATE TABLE mft(INode INTEGER PRIMARY KEY, UserID INT4 NOT NULL, SecurityLevel INT1);");

// Production Rules
doDBAction("DROP TABLE rules;");
doDBAction("CREATE TABLE rules(id INTEGER PRIMARY KEY NOT NULL, variantCount INT4);");
doDBAction("DROP TABLE ruleVariants;");
doDBAction("CREATE TABLE ruleVariants(id INTEGER PRIMARY KEY NOT NULL, variantID INT4);");
doDBAction("DROP TABLE ruleDetails;");
doDBAction("CREATE TABLE ruleDetails(id INTEGER PRIMARY KEY NOT NULL, variantID INT4, Description CHAR(255));");

// Close the database
sqlite3_close_v2(parameters->db);
}

void BuildDB::doDBAction(char *wstmt) {
    // Perform the passed database action on the main database -- makes
    // the Create function cleaner at a small hit on performance.

```

```

char exec_string[2000];
int rc;
sqlite3_stmt *stmt;

////////////////////////////////////

sprintf_s(exec_string, 1000, "%s",wstmt);
rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare |%s| statement\n", rc,wstmt);
    return;
}
sqlite3_step(stmt); // couldn't check result because different statements
                    // have different "normal" results
sqlite3_finalize(stmt);
return;
}

void BuildDB::load() {
    // Purpose: load the low level events and support information

    FILE *dumpfile;                // name of the individual dump files
    TCHAR tpath[MAX_PATH];         // path to pass to processDir

    // Pointers for writing the information in the hash tables
    void *key, *value;
    item_t *witem;
    file_t *wfile;
    application_t *wapplication;
    search_t *wsearch;
    url_t *wurl;
    key_t *wkey;

    // Iterators and Counters
    GHashTableIter htiter;
    int i;

    //////////////////////////////////

```

```

// Initialize variables
// Convert char to tchar -- yes, i know this isn't safe, but it should work
for (i = 0; i < strlen(parameters->path); i++)
    tpath[i] = parameters->path[i]; tpath[i] = 0;

// Process the passed in directory (the indirection is so that subdirectories
// can be processed)
processDir(tpath);
printf("\t\tFinished processing the directory\n");

// Close the dump files that have been populated during the load process
fclose(eventFile);
fclose(activities);
fclose(timeline);
fclose(eventSupport);
fclose(mruList);

// Now, dump the items -- no, I don't have a good reason for doing it to different
//                               but I also don't have a good reason for rewriting it
fopen_s(&dumpfile, ".\\items.out", "w");
g_hash_table_iter_init(&htiter, htItems);
while (g_hash_table_iter_next(&htiter, &key, &value)) {
    witem = (item_t *)value;
    if (witem != NULL)
        fprintf(dumpfile, "%i|i|i|i|i|s\n", witem->itemid, witem->itemtypeid,
            witem->traceType, witem->traceid, witem->sort);
}
fclose(dumpfile);

fopen_s(&dumpfile, ".\\files.out", "w");
g_hash_table_iter_init(&htiter, htFiles);
while (g_hash_table_iter_next(&htiter, &key, &value)) {
    wfile = (file_t *)value;
    if (wfile != NULL)
        fprintf(dumpfile, "%i|i|s|i|i|i\n", wfile->itemid, wfile->hash,
            wfile->fileName, wfile->traceType, wfile->traceid);
}
fclose(dumpfile);

fopen_s(&dumpfile, ".\\applications.out", "w");
g_hash_table_iter_init(&htiter, htApplications);
while (g_hash_table_iter_next(&htiter, &key, &value)) {

```

```

        wapplication = (application_t *)value;
        if (wapplication != NULL)
            fprintf(dumpfile, "%i|s|i|i|i\n", wapplication->itemid,
                    wapplication->applicationName, wapplication->count,
                    wapplication->traceType, wapplication->traceid);
    }
    fclose(dumpfile);

    fopen_s(&dumpfile, ".\\searches.out", "w");
    g_hash_table_iter_init(&htiter, htSearches);
    while (g_hash_table_iter_next(&htiter, &key, &value)) {
        wsearch = (search_t *)value;
        if (wsearch != NULL)
            fprintf(dumpfile, "%i|s|i|i|i\n", wsearch->itemid,
                    wsearch->searchKey, wsearch->application,
                    wsearch->traceType, wsearch->traceid);
    }
    fclose(dumpfile);

    fopen_s(&dumpfile, ".\\urls.out", "w");
    g_hash_table_iter_init(&htiter, htUrls);
    while (g_hash_table_iter_next(&htiter, &key, &value)) {
        wurl = (url_t *)value;
        if (wurl != NULL)
            fprintf(dumpfile, "%i|s|i|i\n",
                    wurl->itemid, wurl->urlName, wurl->traceType, wurl->traceid);
    }
    fclose(dumpfile);

    fopen_s(&dumpfile, ".\\keys.out", "w");
    g_hash_table_iter_init(&htiter, htKeys);
    while (g_hash_table_iter_next(&htiter, &key, &value)) {
        wkey = (key_t *)value;
        if (wkey != NULL)
            fprintf(dumpfile, "%i|s|i|i\n",
                    wkey->itemid, wkey->keyName, wkey->traceType, wkey->traceid);
    }
    fclose(dumpfile);

} // load

int BuildDB::processDir(TCHAR *path) {

```

```

// Purpose: load the low level events and support information for a single director

sqlite3 *chrome, *firefox;

int rc, rc2;

int wlen, wactivity, witemid, wreg_id;
uint32_t wactor, wappexec;
char *witem, *wfriendly, *wguid, *wmac, *wsn;
int i, ctr, ctr2;

int npos;

uint32_t n32time, wseqid, wacount, woid, waction;

HANDLE fh;
TCHAR path2[MAX_PATH], *newPath;
FILE *dumpfile;
char exec_string[1000];

char wfriendly_c[250], wurl_c[250], wguid_c[250], wmac_c[250];
char witem_c[250], wtitle_c[250], stime_c[80], test[200];

TCHAR *wname;
char *wkey, *wvalue, *wtime, *wapplication, *wtemp, *wname_c, *wactor_s;

uint32_t wdConnected, wdCreated, wdAccessed, wdTime, wid, wsize, wstime;
uint32_t wetime, waccessed, wcreated, wmodified, wchanged;
uint8_t wtype, wmanaged, wappid;
uint16_t wactionid;

char *wnamespace, *wfilename, *wfullname, *wstring, *wstring2, *wstring3;

uint8_t isWin32;
uint32_t inode, userid, securityid, parentid, n32createtime, n32modifytime;
uint32_t n32accesstime, n32metatime;

time_t rawtime;
struct tm timeinfo;

char *wsort;

struct tm timeinfo2;

```

```

time_t wtimer;
char showtime[80];

char *wmessage;
char *wperc, *wquote;

char *newstart, *nextStr, *wperc2, *sPos, *ePos, *wsave1;
uint8_t debugevent = 0;
TCHAR *wtempCase, *caseNbr;

////////////////////////////////////

// Convert the directory name int a caseNbr
caseNbr = NULL;
wtempCase = (TCHAR *)malloc(sizeof(TCHAR) * (_tcslen(path) + 2));
_tcsncpy_s(wtempCase, _tcslen(path) + 1, path);
wtempCase[_tcslen(path)] = 0;
if (parameters->debugParms >= 5)
    _tprintf(L"Path = %s\n", (const TCHAR *)wtempCase);
if (_tcsrchr(wtempCase, L'\\') != 0) {
    caseNbr = (TCHAR *)(_tcsrchr(wtempCase, L'\\') + 1);
    if (_tcslen(caseNbr) == 0) {
        _tcsrchr(wtempCase, L'\\')[0] = 0;
        caseNbr = (TCHAR *)(_tcsrchr(wtempCase, L'\\') + 1);
    }
}
else if (_tcsrchr(wtempCase, L'/') != 0) {
    caseNbr = (TCHAR *)(_tcsrchr(wtempCase, L'/') + 1);
    if (_tcslen(caseNbr) == 0) {
        _tcsrchr(wtempCase, L'/')[0] = 0;
        caseNbr = (TCHAR *)(_tcsrchr(wtempCase, L'/') + 1);
    }
}

// For each file in the directory
_tcsncpy_s(path2, _tcslen(path) + 1, path);
_tscat_s(path2, _tcslen(path2) + 3, L"*");
fh = FindFirstFile((LPCTSTR)path2, fd);
if (fh != INVALID_HANDLE_VALUE) {
    do {
        // If it's a subdirectory, call processDir recursively
        if (fd->dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {

```

```

        if (fd->cFileName[0] == '.') continue;
        newPath = (TCHAR *)
            malloc(sizeof(TCHAR) * (_tcslen(path) +
                                    _tcslen(fd->cFileName) + 1 + 1));
        _tcscpy_s(newPath, _tcslen(path) + 1, path);
        _tcscat_s(newPath, _tcslen(path) +
                    _tcslen(fd->cFileName) + 1, fd->cFileName);
        _tcscat_s(newPath, _tcslen(path) +
                    _tcslen(fd->cFileName) + 1 + 1, L"/");
        processDir(newPath);
        continue;
    }

    // Otherwise, get the name of the file -- only process files with a
    // .out extension
    wname = (TCHAR *)malloc(sizeof(TCHAR) * (MAX_PATH + 1));
    _tcscpy_s(wname, _tcslen(path) + 1, path);
    _tcscat_s(wname, _tcslen(path) + _tcslen(fd->cFileName) + 1, fd->cFileName);
    if (_tcsstr(wname, L".out") != NULL) {
        free(wname);
        continue;
    }

    // Report Progress
    time(&rawtime);
    localtime_s(&timeinfo, &rawtime);
    strftime(stime_c, 80, "Current time: %X", &timeinfo);
    printf("\n%s:\t\t", stime_c);
    _tprintf(L"Processing %s\n", wname);

    // Chrome Browse Log
    if (_tcsstr(wname, L"chrome.sql") != NULL) {
        _tprintf(L"\t\t\tAdding Browsing History from %s...\n", wname);

        printf("Opening Chrome DB\n");
        wlen = (int) _tcslen(wname);
        wname_c = (char *)malloc(sizeof(char) * (wlen + 2));
        // Convert TCHAR filename to CHAR actor -- doesn't handle all cases
        for (i = 0; i < wlen; i++) {
            wname_c[i] = wname[i];
        }
        wname_c[wlen] = 0;
        rc = sqlite3_open_v2(wname_c, &chrome, SQLITE_OPEN_READWRITE, NULL);
    }

```

```

if (rc != SQLITE_OK) {
    printf("Error: %s\n", sqlite3_errmsg(chrome));
    return (1);
}

wactor = get_actor(wname);

// Create database statements for the Chrome DB -- can only create these
sqlite3_stmt *stmt_selChromeURL, *stmt_selChromeVisit, *stmt_selChromeDown

sprintf_s(exec_string, 1000, "SELECT ID, url, title FROM urls;");
rc = sqlite3_prepare_v2(chrome, exec_string, 1000, &stmt_selChromeURL, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select url statement\n", rc);
    return (1);
}

sprintf_s(exec_string, 1000, "SELECT visit_time, visit_duration FROM visits;");
rc2 = sqlite3_prepare_v2(chrome, exec_string, 1000, &stmt_selChromeVisit, 0);
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select visits statement\n", rc2);
    return (1);
}

sprintf_s(exec_string, 1000, "SELECT target_path, tab_url, start_time, end_time FROM downloads;");
rc = sqlite3_prepare_v2(chrome, exec_string, 1000, &stmt_selChromeDownload, 0);
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select download statement\n", rc);
    return (1);
}

// Go through all the records in the Chrome DB
rc = sqlite3_step(stmt_selChromeURL);
i = 0;
while (rc == SQLITE_ROW) {

    // Show Progress
    i++;
    if (i % 1000 == 0) printf("URLs processed = %i\n", i);
    wid = sqlite3_column_int(stmt_selChromeURL, 0);

    // Get the web site

```



```

    if (sqlite3_column_text(stmt_selChromeURL, 1) != NULL)
        strncpy_s(witem_c, 249, (const char *)
            sqlite3_column_text(stmt_selChromeURL, 1), 248);
    else
        strncpy_s(witem_c, 249, "N/A", 4);
    if (sqlite3_column_text(stmt_selChromeURL, 2) != NULL)
        strncpy_s(wtitle_c, 249, (const char *)
            sqlite3_column_text(stmt_selChromeURL, 2), 248);
    else
        strncpy_s(witem_c, 249, "N/A", 4);

    // Create an activity for visiting the website
    witemid = get_url(witem_c, S_CHROME_V, wid);
    wactivity = insert_activity(wactor, P_CHROME, A_VISIT,
        witemid, S_CHROME_V, S_CHROME_V, wid);

    // Now go through all the visits at that website and create
    // timeline records for each
    sqlite3_reset(stmt_selChromeVisit);
    rc2 = sqlite3_bind_int(stmt_selChromeVisit, 1, wid);
    rc2 = sqlite3_step(stmt_selChromeVisit);
    while (rc2 == SQLITE_ROW) {
        // Chrome stores time as the # of microseconds since
        // midnight UTC of 1 January 1601.
        wstime = ((uint32_t)(
            (sqlite3_column_int64(stmt_selChromeVisit, 0) -
                116444916000000000) / 1000000));
        wstime += parameters->offsetGMT * 60 * 60;
        wetime = wstime + ((uint32_t)(
            sqlite3_column_int64(stmt_selChromeVisit, 1) /
                1000000));
        insert_timeline(T_DATALOAD, wactivity, NA, R_BETWEEN,
            wstime, wetime, S_CHROME_V, S_CHROME_V, wid, caseNbr);
        rc2 = sqlite3_step(stmt_selChromeVisit);
    }
    rc = sqlite3_step(stmt_selChromeURL);
}
sqlite3_finalize(stmt_selChromeURL);
sqlite3_finalize(stmt_selChromeVisit);

// Now go through the downloads and create event records for them
rc = sqlite3_step(stmt_selChromeDownload);
i = 0;

```

```

while (rc == SQLITE_ROW) {
    if (sqlite3_column_text(stmt_selChromeDownload, 0) != NULL)
        strncpy_s(witem_c, 249, (const char *)
            sqlite3_column_text(stmt_selChromeDownload, 0), 248);
    else
        strncpy_s(witem_c, 249, "N/A", 4);
    if (sqlite3_column_text(stmt_selChromeDownload, 1) != NULL)
        strncpy_s(wurl_c, 249, (const char *)
            sqlite3_column_text(stmt_selChromeDownload, 1), 248);
    else
        strncpy_s(wurl_c, 249, "N/A", 4);
    wid = get_url(wurl_c, S_CHROME_V, S_CHROME_V);
    witemid = get_file(witem_c, S_CHROME_V, wid);
    wactivity = insert_activity(wactor, P_CHROME, A_DOWNLOAD,
        witemid, S_CHROME_V, S_CHROME_V, wid);

    wstime = ((uint32_t)((sqlite3_column_int64(
        stmt_selChromeDownload, 2) - 116444916000000000) / 1000000));
    wstime += parameters->offsetGMT * 60 * 60;
    wetime = ((uint32_t)((sqlite3_column_int64(
        stmt_selChromeDownload, 2) - 116444916000000000) / 1000000));
    wetime += parameters->offsetGMT * 60 * 60;
    insert_timeline(T_DATALOAD, wactivity, NA, R_BETWEEN, wstime,
        wetime, S_CHROME_D, S_CHROME_D, wid, caseNbr);
    rc = sqlite3_step(stmt_selChromeDownload);
}
sqlite3_finalize(stmt_selChromeDownload);

free(wname_c);
printf("Closing Chrome DB\n");
sqlite3_close(chrome);
continue;
}

// Firefox Browser Log
if (_tcsstr(wname, L"firefox.sql") != NULL) {
    _tprintf(L"\t\t\tAdding Browsing History from %s...\n", wname);

    // open the Firefox DB
    printf("Opening Firefox DB\n");
    wlen = (int) _tcslen(wname);
    wname_c = (char *)malloc(sizeof(char) * (wlen + 1));
    // Convert TCHAR filename to CHAR actor -- doesn't handle all cases

```

```

for (i = 0; i < wlen; i++) wname_c[i] = wname[i];
wname_c[wlen] = 0;
rc = sqlite3_open_v2(wname_c, &firefox, SQLITE_OPEN_READWRITE, NULL);
if (rc != SQLITE_OK) {
    printf("Error: %s\n", sqlite3_errmsg(firefox));
    return (1);
}

wactor = get_actor(wname);

// Create database statements for the Firefox DB -- can only create these
// statements if there is a Firefox DB
sqlite3_stmt *stmt_selFirefoxURL, *stmt_selFirefoxVisit, *stmt_selFirefoxDownl

sprintf_s(exec_string, 1000, "SELECT ID, url FROM moz_places;");
rc = sqlite3_prepare_v2(firefox, exec_string, 1000, &stmt_selFirefoxURL,
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select firefox url statement\n", r
    return (1);
}

sprintf_s(exec_string, 1000, "SELECT visit_date FROM moz_historyvisits WH
rc2 = sqlite3_prepare_v2(firefox, exec_string, 1000, &stmt_selFirefoxVisi
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select visits statement\n", rc2);
    return (1);
}

sprintf_s(exec_string, 1000, "SELECT content, dateAdded, lastModified FRO
rc = sqlite3_prepare_v2(firefox, exec_string, 1000, &stmt_selFirefoxDownl
if (rc != SQLITE_OK) {
    printf("Error: %i. couldn't prepare select download statement\n", rc);
    return (1);
}

// Go through all the records in the Firefox DB
rc = sqlite3_step(stmt_selFirefoxURL);
i = 0;
while (rc == SQLITE_ROW) {

    // Show progress
    i++;
    if (i % 1000 == 0) printf("URLs processed = %i\n", i);

```

```

wid = sqlite3_column_int(stmt_selFirefoxURL, 0);

// Get the URL visited
if (sqlite3_column_text(stmt_selFirefoxURL, 1) != NULL)
    strncpy_s(witem_c, 249, (const char *)
        sqlite3_column_text(stmt_selFirefoxURL, 1), 248);
else
    strncpy_s(witem_c, 249, "N/A", 4);

// Create an activity for the visit
witemid = get_url(witem_c, S_FIREFOX_V, wid);
wactivity = insert_activity(wactor, P_FIREFOX, A_VISIT, witemid,
    S_FIREFOX_V, S_FIREFOX_V, wid);

// For each visit, create an event record
sqlite3_reset(stmt_selFirefoxVisit);
rc2 = sqlite3_bind_int(stmt_selFirefoxVisit, 1, wid);
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't bind ID\n", rc2);
    return (1);
}
rc2 = sqlite3_step(stmt_selFirefoxVisit);
while (rc2 == SQLITE_ROW) {
    // Firefox stores time as the # of microseconds since
    // midnight UTC of 1 January 1970.
    wstime = ((uint32_t)(sqlite3_column_int64(
        stmt_selFirefoxVisit, 0) / 1000000));
    wstime += parameters->offsetGMT * 60 * 60;
    insert_timeline(T_DATALOAD, wactivity, NA, R_BETWEEN, wstime,
        wstime, S_FIREFOX_V, S_FIREFOX_V, wid, caseNbr);
    rc2 = sqlite3_step(stmt_selFirefoxVisit);
}

// For each download from that URL, create an event record
sqlite3_reset(stmt_selFirefoxDownload);
rc2 = sqlite3_bind_int(stmt_selFirefoxDownload, 1, wid);
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't bind ID\n", rc2);
    return (1);
}
rc2 = sqlite3_step(stmt_selFirefoxDownload);
while (rc2 == SQLITE_ROW) {
    if (sqlite3_column_text(stmt_selFirefoxDownload, 0) != NULL)

```

```

        strncpy_s(witem_c, 249, (const char *)
            sqlite3_column_text(stmt_selFirefoxDownload, 0), 248);
    else
        strncpy_s(witem_c, 249, "N/A", 4);

    wid = get_url(wurl_c, S_FIREFOX_V, S_FIREFOX_V);
    witemid = get_file(witem_c, S_FIREFOX_V, wid);
    wactivity = insert_activity(wactor, P_FIREFOX, A_DOWNLOAD,
        witemid, S_FIREFOX_V, S_FIREFOX_V, wid);

    // Firefox stores time as the # of microseconds since
    // midnight UTC of 1 January 1970.
    wstime = ((uint32_t)(sqlite3_column_int64(
        stmt_selFirefoxDownload, 1) / 1000000));
    wstime += parameters->offsetGMT * 60 * 60;
    insert_timeline(T_DATALOAD, wactivity, NA, R_BETWEEN, wstime,
        wstime, S_FIREFOX_V, S_FIREFOX_V, wid, caseNbr);
    rc2 = sqlite3_step(stmt_selFirefoxDownload);
}
rc = sqlite3_step(stmt_selFirefoxURL);

}
sqlite3_finalize(stmt_selFirefoxURL);
sqlite3_finalize(stmt_selFirefoxVisit);
sqlite3_finalize(stmt_selFirefoxDownload);

free(wname_c);
printf("Closing Firefox DB\n");
sqlite3_close(firefox);
continue;
}

// If it was a Chrome or Firefox parameters->db, it got opened above;
// otherwise it's getting opened now
if (_tfopen_s(&dumpfile, wname, L"rb")) {
    _tprintf(L"Dump file %s can't be opened. Sorry.\n", wname);
    continue;
}

// Event Logs
if (_tcsstr(wname, L"eventlog.txt") != NULL) {
    _tprintf(L"\t\t\tAdding Event Logs from %s...\n", wname);
    sqlite3_reset(stmt_resetEM);
}

```

```

sqlite3_step(stmt_resetEM);
sqlite3_reset(stmt_resetEM2);
sqlite3_step(stmt_resetEM2);

// Remove headings
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; free(wstring);
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; free(wstring);
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; free(wstring);
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; free(wstring);
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; free(wstring);

// For each record in the event log
ctr2 = 0;
while (1) {

    // Show Progress
    ctr2++;
    if (ctr2 % 100 == 0) {
        time(&wtimer); localtime_s(&timeinfo2, &wtimer);
        strftime(showtime, 80, "Current time: %X", &timeinfo2);
        printf("Processed %i records\t\t%s\n", ctr2, showtime);
    }

    // Parse the time
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break;
    n32time = utilities->stringToTime(wstring);
    free(wstring);

    // Get the application
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break;
    woid = atoi(wstring); free(wstring); // EventID
    wapplication = utilities->firead(dumpfile, '\t', 0);
    if (wapplication == NULL) break; // application
    utilities->strtoupperIP(wapplication);
}

```

```

// A single exception to the normal rule
if (utilities->stringSearchU("vmauthd", wapplication) >= 0) {
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, '\t', 0);
    if (wstring == NULL) break; free(wstring);
    continue;
}

wappexec = get_appexec(wapplication, S_EVENT, woid);

// Get the actor
wactor_s = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; wactor = get_actor2(wactor_s); // Actor
wstring = utilities->firead(dumpfile, '\t', 0);
if (wstring == NULL) break; // message

// Now, lookup the event message in the event message database
strcpy_s(test, strlen(wapplication) + 1, wapplication);
test[strlen(wapplication) + 1] = 0;
sqlite3_reset(stmt_selEMs);
sqlite3_bind_int(stmt_selEMs, 1, woid);
sqlite3_bind_text(stmt_selEMs, 2, test, -1, SQLITE_TRANSIENT);
rc = sqlite3_step(stmt_selEMs);
ctr = (rc == SQLITE_ROW ?
        sqlite3_column_int(stmt_selEMs, 0) : 0);
/// If it doesn't match id1, try id2
if (rc != SQLITE_ROW || ctr != 1) {
    sqlite3_reset(stmt_selEMs2);
    sqlite3_bind_int(stmt_selEMs2, 1, woid);
    sqlite3_bind_text(stmt_selEMs2, 2, test, -1, SQLITE_TRANSIENT);
    rc = sqlite3_step(stmt_selEMs2);
    ctr = (rc == SQLITE_ROW ?
            sqlite3_column_int(stmt_selEMs2, 0) : 0);
}

// If we found the message and there's only one message with
// that id and it's a normal message
if ((ctr == 1) &&

```

```

(strstr(wstring, "The description for Event ID") == 0)) {
wmessage = NULL;
// Get the message
sqlite3_reset(stmt_selEM);
sqlite3_bind_int(stmt_selEM, 1, woid);
sqlite3_bind_text(stmt_selEM, 2, test, -1, SQLITE_TRANSIENT);
rc = sqlite3_step(stmt_selEM);
if (rc == SQLITE_ROW) {
    // Get the message and id and add them to action table
    wmessage = (char *)sqlite3_column_text(stmt_selEM, 0);
    waction = sqlite3_column_int(stmt_selEM, 1);
    if (waction == A_EVENT) {
        sqlite3_reset(stmt_insActions);
        sqlite3_bind_text(stmt_insActions, 1,
                        wmessage, -1, SQLITE_TRANSIENT);
        rc = sqlite3_step(stmt_insActions);
        waction = last_insert(parameters->db);
        sqlite3_reset(stmt_updEM);
        sqlite3_bind_int(stmt_updEM, 1, woid);
        sqlite3_bind_text(stmt_updEM, 2, test, -1,
                        SQLITE_TRANSIENT);
        sqlite3_bind_int(stmt_updEM, 3, waction);
        rc = sqlite3_step(stmt_updEM);
    }
}
else { // if it doesn't match id1, try id2
    sqlite3_reset(stmt_selEM2);
    sqlite3_bind_int(stmt_selEM2, 1, woid);
    sqlite3_bind_text(stmt_selEM2, 2, test, -1, SQLITE_TRANSIENT);
    rc = sqlite3_step(stmt_selEM2);
    if (rc == SQLITE_ROW) {
        // Get the message and id and add them to action table
        wmessage = (char *)sqlite3_column_text(stmt_selEM2, 0);
        waction = sqlite3_column_int(stmt_selEM2, 1);
        if (waction == A_EVENT) {
            sqlite3_reset(stmt_insActions);
            sqlite3_bind_text(stmt_insActions, 1,
                            wmessage, -1, SQLITE_TRANSIENT);
            rc = sqlite3_step(stmt_insActions);
            waction = last_insert(parameters->db);
            sqlite3_reset(stmt_updEM2);
            sqlite3_bind_int(stmt_updEM2, 1, woid);
            sqlite3_bind_text(stmt_updEM2, 2, test, -1,

```



```

        SQLITE_TRANSIENT);
        sqlite3_bind_int(stmt_updEM2, 3, waction);
        sqlite3_step(stmt_updEM2);
    }
}

// If you found the message in the message database (
// this should always be true since you found it before
if (rc == SQLITE_ROW) {
    // Add the event, activity and timeline records

    wid = insert_event(woid, n32time, wappexec, wactor,
        waction, wstring);
    wactivity = insert_activity(wactor, wappexec, waction,
        witemid, S_EVENT, S_EVENT, wid);
    insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
        n32time, NA, S_EVENT, S_EVENT, wid, caseNbr);

    // now work on the event support records
    // I've left this in even though I'm not currently
    // saving the event support records
    // TODO: start adding eventSupport records
    for (i = (int)strlen(wmessage) - 1; i > 0; i--)
        if ((wmessage[i] > 32) && (wmessage[i] < 128)) break;
    wmessage[i + 1] = 0;
    if (debugevent) printf("wmessage = %s\n", wmessage);
    if (debugevent) printf("wstring = %s\n", wstring);
    wperc = strchr(wmessage, '%');
    wquote = strchr(wmessage, 34);
    if ((wperc != NULL) && (wquote != NULL) && (wquote == wperc - 1))
        wperc = wquote;
    ctr = 0;
    wsavel = wstring;
    while ((wperc != NULL) && (strlen(wstring) > 0)) {
        ctr++;
        i = wperc - wmessage + 1;
        sPos = wstring + i - 1;
        if (debugevent)
            printf("Variable %i: starting location: %i\n", ctr, i - 1)
        while ((i < strlen(wmessage)) &&
            (((wmessage[i] >= 48) && (wmessage[i] <= 57)) ||
            (wmessage[i] == '%') || (wmessage[i] == 'n') ||

```

```

        (wmessage[i] == 34))) {
    i++;
}
newstart = wmessage + i;
if (debugevent) printf("newstart = %s\n", newstart);
wperc2 = strchr(newstart, '%');
wquote = strchr(newstart, 34);
if ((wperc2 != NULL) && (wquote != NULL) &&
    (wquote == wperc2 - 1)) wperc2 = wquote;
if (wperc2 != NULL) {
    nextStr = (char *)malloc(
        sizeof(char) * (wperc2 - newstart + 1));
    for (i = 0; i < wperc2 - newstart; i++)
        nextStr[i] = newstart[i]; nextStr[i] = 0;
}
else {
    nextStr = (char *)
        malloc(sizeof(char) * (strlen(newstart) + 1));
    for (i = 0; i < strlen(newstart); i++)
        nextStr[i] = newstart[i]; nextStr[i] = 0;
}
if (debugevent)
    printf("nextString = %s    length = %zi\n",
        nextStr, strlen(nextStr));
if (strlen(nextStr) == 0)
    ePos = NULL;
else {
    ePos = strstr(wstring, nextStr);
    if (debugevent)
        printf("epos = %llx    %s\n", (uint64_t)ePos, ePos);
    free(nextStr);
}
if (ePos == NULL) ePos = sPos + strlen(sPos);
if (debugevent)
    printf("spos = %llx epos = %llx    \n",
        (uint64_t)sPos, (uint64_t)ePos);
if (ePos <= sPos) {
    ePos = sPos;
    witem = (char *)malloc(sizeof(char) * 2);
    witem[0] = 0;
    witem[1] = 0;
}
else {

```

```

        witem = (char *)malloc(sizeof(char) * (ePos - sPos + 1));
        for (i = 0; i < ePos - sPos; i++)
            witem[i] = wstring[(sPos - wstring) + i];
        witem[i] = 0;
    }

    // this is where you'd put the insert_eventSupport
    // insert_event_support(parameters->db, wactivity, ???, witem);
    free(witem);

    wstring = sPos + (ePos - sPos);
    wmessage = newstart;

    wperc = strchr(wmessage, '%');
    wquote = strchr(wmessage, 34);
    if (debugevent)
        printf("wquote = %llx    wperc = %llx\n",
            (uint64_t)wquote, (uint64_t)wperc);
    if ((wperc != NULL) && (wquote != NULL) &&
        (wquote == wperc - 1)) wperc = wquote;
    if (debugevent)
        printf("wmessage (%zi) = %s\n", strlen(wmessage), wmessage);
    if (debugevent)
        printf("wstring (%zi) = %s\n", strlen(wstring), wstring);
    }
    wstring = wsave1;
}

// Either there's duplicates of the id or it says
// "The description for event id is missing"
else {
    if ((ctr > 1) && strstr(wstring, "The description for Event ID") ==
        printf("Too many (%i):    %i: %s          %s\n", ctr, woid, wapplica
    // If we want to handle these, we need to do something manually
    // (and probably make sure each id/app only shows up once
    else if (strstr(wstring, "The description for Event ID") == 0)
        printf("**MISSING** (%i) %i:  application = |%s|    %s\n",
            ctr, woid, wapplication, wstring);
    }

    free(wactor_s);
    free(wstring);

```

```

    }
}

// File Times
else if (_tcsstr(wname, L"_mactime") != NULL) { // File System
    _tprintf(L"\t\t\tAdding Mac Times from %s...\n", wname);
    i = 0;

    // For each record
    while (1) {
        i++;
        // Show progress
        if (i % 10000 == 0) printf("i = %i\n", i);

        // Parse the record
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); // MD5
        witem = utilities->firead(dumpfile, '|', 1);
        if (witem == NULL) break; // path/name
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); //inode
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); //mode as string (permissions)
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); // uid
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); // gid
        wstring = utilities->firead(dumpfile, '|', 1);
        if (wstring == NULL) break; free(wstring); // size
        wtemp = utilities->firead(dumpfile, '|', 1);
        if (wtemp == NULL) break;
        waccessed = atoi(wtemp); free(wtemp);
        wtemp = utilities->firead(dumpfile, '|', 1);
        if (wtemp == NULL) break;
        wmodified = atoi(wtemp); free(wtemp);
        wtemp = utilities->firead(dumpfile, '|', 1);
        if (wtemp == NULL) break;
        wchanged = atoi(wtemp); free(wtemp);
        wtemp = utilities->firead(dumpfile, '|', 1);
        if (wtemp == NULL) break;
        wcreated = atoi(wtemp); free(wtemp);

        // Get the itemid for the file

```

```

        if (utilities->stringSearch("N/A", witem) == -1) {
            witemid = get_file(witem, S_TSK, NA);
        }
        else {
            witemid = NA;
        }

        // Create the event
        if (wcreated > 0) {
            wactivity = insert_activity(NA, NA, A_CREATE, witemid,
                                       S_TSK, S_TSK, NA);
            insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                           wcreated, NA, S_TSK, S_TSK, NA, caseNbr);
        }
        if (wmodified > 0) {
            wactivity = insert_activity(NA, NA, A_MODIFY, witemid,
                                       S_TSK, S_TSK, NA);
            insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                           wmodified, NA, S_TSK, S_TSK, NA, caseNbr);
        }
        if (waccesssed > 0) {
            wactivity = insert_activity(NA, NA, A_ACCESS, witemid,
                                       S_TSK, S_TSK, NA);
            insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                           waccesssed, NA, S_TSK, S_TSK, NA, caseNbr);
        }
        if (wchanged > 0) {
            wactivity = insert_activity(NA, NA, A_CHANGE, witemid,
                                       S_TSK, S_TSK, NA);
            insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                           wchanged, NA, S_TSK, S_TSK, NA, caseNbr);
        }
        free(witem);
    }
}

// File Times -- two different formats (only one will exist in a single dump)
else if (_tcsstr(wname, L"ntfsinfo.txt") != NULL) { // MFT
    _tprintf(L"\t\t\t\t\tCreating MFT from %s...\n", wname);
    _tfopen_s(&dumpfile, wname, L"rb");

    if (dumpfile == NULL) { printf("    failed to open file\n"); continue; }
    wnnamespace = NULL;

```

```

wfilename = NULL;
wfullname = NULL;
isWin32 = 0;
i = 0;
inode = userid = securityid = parentid = 0;
n32createtime = n32modifytime = n32accesstime = n32metatime = 0;

// For each record
while (1) {
    i++;

    // Show progress
    if (i % 100000 == 0) printf("i = %i\n", i);

    // Parse the current line -- note: different lines have
    // different formats
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break;

    // filename -- will have the four dates, create four
    // event records for it
    if (wstring[0] == '/') { // Need to handle some (but not all) file names
        if (utilities->filastChar() == 32) {
            wstring2 = utilities->firead_nl(dumpfile, ' ');
            wstring3 = (char *)malloc(sizeof(char) *
                (strlen(wstring) + strlen(wstring2) + 3));
            strcpy_s(wstring3, strlen(wstring) + 1, wstring);
            strcat_s(wstring3, strlen(wstring) + 2, " ");
            strcat_s(wstring3, strlen(wstring) +
                strlen(wstring2) + 2, wstring2);
            wstring3[strlen(wstring) + strlen(wstring2) + 1] = 0;
            free(wstring);
            free(wstring2);
            wstring = wstring3;
        }
    }
    if ((inode != 0) && (wfilename != NULL) && (isWin32)) { // Create record
        if (parameters->debugParms >= 3)
            printf("Inserting %i (%s)\n", inode, wfullname);
        if (utilities->stringSearch("N/A", witem) == -1) {
            witemid = get_file(wfullname, S_TSK, NA);
        }
        else {
            witemid = NA;
        }
    }
}

```

```

    }
    if (n32createtime > 0) {
        wactivity = insert_activity(NA, NA, A_CREATE, witemid,
                                    S_TSK, S_TSK, NA);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                        n32createtime, NA, S_TSK, S_TSK, NA, caseNbr);
    }
    if (n32modifytime > 0) {
        wactivity = insert_activity(NA, NA, A_MODIFY, witemid,
                                    S_TSK, S_TSK, NA);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                        n32modifytime, NA, S_TSK, S_TSK, NA, caseNbr);
    }
    if (n32accesstime > 0) {
        wactivity = insert_activity(NA, NA, A_ACCESS, witemid,
                                    S_TSK, S_TSK, NA);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                        n32accesstime, NA, S_TSK, S_TSK, NA, caseNbr);
    }
    if (n32metatime > 0) {
        wactivity = insert_activity(NA, NA, A_CHANGE, witemid,
                                    S_TSK, S_TSK, NA);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                        n32metatime, NA, S_TSK, S_TSK, NA, caseNbr);
    }

    inode = userid = securityid = parentid = 0;
    n32createtime = n32modifytime = n32accesstime = n32metatime = 0;
    if (wnamespace != NULL) free(wnamespace);
    if (wfilename != NULL) free(wfilename);
    if (wfullname != NULL) free(wfullname);
    wnamespace = wfilename = wfullname = NULL;
    isWin32 = 0;
    inode = userid = securityid = parentid = 0;
    n32createtime = n32modifytime = n32accesstime = n32metatime = 0;
    free(wnamespace); free(wfilename);
    wnamespace = NULL; wfilename = NULL;
}
wfullname = wstring;
if (parameters->debugParms >= 3)
    printf("Setting full filename to %s    lastChar = %i\n",
          wstring, utilities->filastChar());
}

```

```

// Dumping inode
else if (strstr(wstring, "Dumping") != NULL) {
    free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; inode = atoi(wstring); free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    if (parameters->debugParms >= 3)
        printf("Setting iNode to %i\n", inode);
}

// User that owns the file
else if (strstr(wstring, "User") != NULL) {
    free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; userid = atoi(wstring); free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    if (parameters->debugParms >= 3)
        printf("Setting User to %i\n", userid);
}

// Security id
else if (strstr(wstring, "Security") != NULL) {
    free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; securityid = atoi(wstring); free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    if (parameters->debugParms >= 3)
        printf("Setting security to %i\n", securityid);
}

// Parent?
else if (strstr(wstring, "Parent") != NULL) {
    free(wstring);

```



```

wstring = utilities->firead(dumpfile, ' ', 1);
if (wstring == NULL) break; free(wstring);
wstring = utilities->firead(dumpfile, ' ', 1);
if (wstring == NULL) break; parentid = atoi(wstring); free(wstring);
wstring = utilities->firead(dumpfile, ' ', 1);
if (wstring == NULL) break; free(wstring);
if (parameters->debugParms >= 3)
    printf("Setting parent to %i\n", parentid);
}

// Namespace
else if (strstr(wstring, "Namespace:") != NULL) {
    free(wstring);
    if (isWin32 == 0) {
        wnamespace = utilities->firead_nl(dumpfile, ' ');
        if (wnamespace == NULL) break;
        if (strstr(wnamespace, "Win32") != NULL) isWin32 = 1;
    }
    else {
        wstring = utilities->firead_nl(dumpfile, ' ');
        if (wstring == NULL) break; free(wstring);
    }
    if (parameters->debugParms >= 3)
        printf("Setting Namespace to |%s|   win32 = %i\n", wnamespace, isWin32);
}

// Name of File
else if (strstr(wstring, "Filename:") != NULL) {
    free(wstring);
    if ((isWin32 == 1) && (wfilename == NULL)) {
        wfilename = utilities->firead_nl(dumpfile, ' ');
        if (wfilename == NULL) break;
    }
    else {
        wstring = utilities->firead_nl(dumpfile, ' ');
        if (wstring == NULL) break;
        free(wstring);
    }
    if (parameters->debugParms >= 3)
        printf("Setting filename to |%s|\n", wfilename);
}

// Additional date changes

```

```

else if (strstr(wstring, "File") != NULL) {
    free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break;
    if (strstr(wstring, "Creation") != NULL) {
        free(wstring);
        wstring = utilities->firead(dumpfile, ' ', 1);
        if (wstring == NULL) break; free(wstring);
        wstring = utilities->firead_nl(dumpfile, ' ');
        if (wstring == NULL) break;
        n32createtime = utilities->stringToTime(wstring); // Time
        if (parameters->debugParms >= 3)
            printf("Setting file creation time to %i |%s|\n", n32createti
        free(wstring);
    }
    else if (strstr(wstring, "Altered") != NULL) {
        free(wstring);
        wstring = utilities->firead(dumpfile, ' ', 1);
        if (wstring == NULL) break; free(wstring);
        wstring = utilities->firead_nl(dumpfile, ' ');
        if (wstring == NULL) break;
        n32modifytime = utilities->stringToTime(wstring); // Time
        if (parameters->debugParms >= 3)
            printf("Setting file modify time to %i |%s|\n",
                n32modifytime, wstring);
        free(wstring);
    }
}
else if (strstr(wstring, "MFT") != NULL) {
    free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead(dumpfile, ' ', 1);
    if (wstring == NULL) break; free(wstring);
    wstring = utilities->firead_nl(dumpfile, ' ');
    if (wstring == NULL) break;
    n32metatime = utilities->stringToTime(wstring); // Time
    if (parameters->debugParms >= 3)
        printf("Setting file meta change time to %i |%s|\n", n32metatime
    free(wstring);
}
else if (strstr(wstring, "Last") != NULL) {
    free(wstring);

```

```

        wstring = utilities->firead(dumpfile, ' ', 1);
        if (wstring == NULL) break; free(wstring);
        wstring = utilities->firead(dumpfile, ' ', 1);
        if (wstring == NULL) break; free(wstring);
        wstring = utilities->firead_nl(dumpfile, ' ');
        if (wstring == NULL) break;
        n32accesstime = utilities->stringToTime(wstring); // Time
        if (parameters->debugParms >= 3)
            printf("Setting file access time to %i  |%s|\n", n32accesstime,
                free(wstring);
    }
    else {
        if (parameters->debugParms >= 3) printf("str = %s\n", wstring);
        free(wstring);
    }
}

}

// Registry records
else if (_tcsstr(wname, L"_reg.txt") != NULL) { // Registry
    _tprintf(L"\t\t\tAdding Registry Values from %s...\n", wname);

    // Pull the user name from the filename
    wactor = get_actor(wname);

    // For each record
    while (1) {

        // Get key, value, and last update time
        wkey = utilities->firead(dumpfile, '\t', 1);
        if (wkey == NULL) break;
        wvalue = utilities->firead(dumpfile, '\t', 1);
        if (wvalue == NULL) break;
        do {
            wtime = utilities->firead(dumpfile, '\t', 1);
        } while ((wtime != NULL) && (!(utilities->isnum(wtime))));
        if (wtime == NULL) break;

        wsize = (unsigned int)strlen(wvalue);
        n32time = atoi(wtime);

        //////////////////////////////////////

```

```

// First, place it in the registry table //
////////////////////////////////////////
wreg_id = insert_registry(wkey, (wsize <= 250 ? "CHAR" : "TEXT"),
                          wvalue, n32time);

////////////////////////////////////////
// Second, if appropriate, put it in a key-specific table //
////////////////////////////////////////
if (utilities->stringSearch(wkey, "**TIME DATA ONLY**/") > -1) {
    witem = (char *)malloc(sizeof(char) * (strlen(wkey) - 18));
    for (i = 0; i < strlen(wkey) - 19; i++)
        witem[i] = wkey[i + 19];
    witem[i] = 0;
    witemid = get_key(witem, S_REGISTRY, wreg_id);

    wactivity = insert_activity(wactor, NA, A_REG, witemid,
                               S_REGISTRY, S_REGISTRY, wreg_id);
    insert_timeline(T_DATALOAD, wactivity, NA, R_AT, n32time,
                   NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
    free(witem);
}
else if ((utilities->stringSearch(wkey,
    "Microsoft/Windows Portable Devices/Devices") > 0)
|| ((utilities->stringSearch(wkey,
    "Control/DeviceClasses/53F56307-B6BF-11D0-94F2-00A0C91EFB8B")
    && (utilities->stringSearch(wkey, "DeviceInstance") > 0))
|| ((utilities->stringSearch(wkey,
    "Control/DeviceClasses/53F5630D-B6BF-11D0-94F2-00A0C91EFB8B")
    && (utilities->stringSearch(wkey, "DeviceInstance") > 0))
|| ((utilities->stringSearch(wkey,
    "Enum/USB") > 0) //USB Removable Media - GUID Keys for Volume
    && (utilities->stringSearch(wkey, "Service") > 0))
|| (utilities->stringSearch(wkey, "MountedDevices") > 0) //Mounted
|| (utilities->stringSearch(wkey, "MountPoints2") > 0) //Mounted De
) {
    // Strip off the device#, serial#, and friendly name
    witem = utilities->strip(wvalue, ' ', 0);
    if (witem == NULL) witem = utilities->na_char();
    wsn = utilities->strip(wvalue, ' ', 1);
    if (wsn == NULL) wsn = utilities->na_char();
    wfriendly = utilities->stripLabel(wvalue, "NAME:", ' ');
    if (wfriendly == NULL) wfriendly = utilities->na_char();
}

```

```

wguid = utilities->stripLabel(wvalue, "GUID:", ' ');
if (wguid == NULL) wguid = utilities->na_char();
wmac = utilities->stripLabel(wvalue, "MAC:", ' ');
if (wmac == NULL) wmac = utilities->na_char();

// Check if the device already exists
// If a device name was specified,
if (utilities->stringSearch("N/A", witem) == -1) {
    // See if the device already exists
    sqlite3_reset(stmt_selDevice);
    rc = sqlite3_bind_text(stmt_selDevice, 1, witem, -1,
        SQLITE_TRANSIENT);
    rc = sqlite3_bind_text(stmt_selDevice, 2, wsn, -1,
        SQLITE_TRANSIENT);
    rc = sqlite3_step(stmt_selDevice);
    // If it already exists, update the friendly name and GUID
    // if they were blank
    if (rc == SQLITE_ROW) { // read row
        strncpy_s(wfriendly_c, 249, (const char *)
            sqlite3_column_text(stmt_selDevice, 0), 248);
        if (utilities->stringSearch("N/A", wfriendly_c) > 0)
            strncpy_s(wfriendly_c, 249, wfriendly, min(strlen(wfriendly), 248));
        strncpy_s(wguid_c, 249, (const char *)
            sqlite3_column_text(stmt_selDevice, 1), 248);
        if (utilities->stringSearch("N/A", wguid_c) > 0)
            strncpy_s(wguid_c, 249, wguid, min(strlen(wguid), 248));
        strncpy_s(wmac_c, 249, (const char *)
            sqlite3_column_text(stmt_selDevice, 2), 248);
        if (utilities->stringSearch("N/A", wmac_c) > 0)
            strncpy_s(wmac_c, 249, wmac, min(strlen(wmac), 248));
        witemid = sqlite3_column_int(stmt_selDevice, 3);

        rc = sqlite3_bind_text(stmt_updDevice, 1,
            wfriendly_c, -1, SQLITE_TRANSIENT); // FriendlyName
        rc = sqlite3_bind_text(stmt_updDevice, 2,
            wguid_c, -1, SQLITE_TRANSIENT); // GUID
        rc = sqlite3_bind_text(stmt_updDevice, 3,
            wmac_c, -1, SQLITE_TRANSIENT); // MAC
        rc = sqlite3_bind_text(stmt_updDevice, 4,
            witem, -1, SQLITE_TRANSIENT); // Device
        rc = sqlite3_bind_text(stmt_updDevice, 5,
            wsn, -1, SQLITE_TRANSIENT); // Serial Number
        rc = sqlite3_step(stmt_updDevice);
    }
}

```

```

        if (rc != SQLITE_DONE) {
            printf("Error: %i.  couldn't execute insert items\n", rc);
            return (1);
        }
    }
}
else {
    // Add the device to the item table
    wsort = (char *)malloc(sizeof(char) * (strlen(witem) + 2));
    strcpy_s(wsort, strlen(witem) + 1, witem);
    witemid = insert_item(I_DEVICE, S_REGISTRY, wreg_id, wsort);
    // Add the device to the device table
    rc = sqlite3_bind_int(stmt_insDevice, 1, witemid); // ItemID
    rc = sqlite3_bind_text(stmt_insDevice, 2,
                           witem, -1, SQLITE_TRANSIENT); // DeviceID
    rc = sqlite3_bind_text(stmt_insDevice, 3,
                           wsn, -1, SQLITE_TRANSIENT); // SN
    rc = sqlite3_bind_text(stmt_insDevice, 4,
                           wfriendly, -1, SQLITE_TRANSIENT); // Friendly
    rc = sqlite3_bind_text(stmt_insDevice, 5,
                           wguid, -1, SQLITE_TRANSIENT); // GUID
    rc = sqlite3_bind_text(stmt_insDevice, 6,
                           wmac, -1, SQLITE_TRANSIENT); // MAC Address
    rc = sqlite3_bind_int(stmt_insDevice, 7,
                           S_REGISTRY); // Trace Type
    rc = sqlite3_bind_int(stmt_insDevice, 8,
                           wreg_id); // Trace ID
    rc = sqlite3_step(stmt_insDevice);
    if (rc != SQLITE_DONE) {
        printf("Error: %i.  couldn't execute insert devices\n", rc);
        return (1);
    }
}
// Otherwise, if the device name wasn't specified
}
else {
    // See if you can find it via the GUID and MAC
    rc = sqlite3_bind_text(stmt_selDevice2, 1, wguid, -1,
                           SQLITE_TRANSIENT);
    rc = sqlite3_bind_text(stmt_selDevice2, 2, wmac, -1,
                           SQLITE_TRANSIENT);
    rc = sqlite3_step(stmt_selDevice2);
    // If you can, use that item ID
    if (rc == SQLITE_ROW) { // read row

```

```

        witemid = sqlite3_column_int(stmt_selDevice2, 0);
        // If you can't add the device
    }
    else {
        // Add the device to the item table
        wsort = (char *)malloc(sizeof(char) * (strlen(witem) + 1));
        strcpy_s(wsort, strlen(witem) + 1, witem);
        witemid = insert_item(I_DEVICE, S_REGISTRY, wreg_id, wsort);
        // Add the device to the device table
        rc = sqlite3_bind_int(stmt_insDevice2, 1, witemid); // ItemID
        rc = sqlite3_bind_text(stmt_insDevice2, 2,
                                witem, -1, SQLITE_TRANSIENT); // DeviceID
        rc = sqlite3_bind_text(stmt_insDevice2, 3,
                                wsn, -1, SQLITE_TRANSIENT); // SN
        rc = sqlite3_bind_text(stmt_insDevice2, 4,
                                wfriendly, -1, SQLITE_TRANSIENT); // Friendly
        rc = sqlite3_bind_text(stmt_insDevice2, 5,
                                wguid, -1, SQLITE_TRANSIENT); // GUID
        rc = sqlite3_bind_text(stmt_insDevice2, 6,
                                wmac, -1, SQLITE_TRANSIENT); // MAC Address
        rc = sqlite3_bind_int(stmt_insDevice2, 7, S_REGISTRY); // Tra
        rc = sqlite3_bind_int(stmt_insDevice2, 8, wreg_id); // Trace
        rc = sqlite3_step(stmt_insDevice2);
    }
}

wactivity = insert_activity(wactor, NA, A_CONNECT,
                            witemid, S_REGISTRY, S_REGISTRY, wreg_id);
insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                n32time, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
free(witem);
free(wsn);
free(wfriendly);
free(wguid);
free(wmac);
}
else if (utilities->stringSearch(wkey,
    "Microsoft/Windows NT/CurrentVersion/NetworkList") > 0) {
    witem = utilities->stripLabel(wvalue, "NAME:", '~');
    if (witem == NULL) witem = utilities->na_char();
    npos = utilities->stringSearch(wvalue, "MANAGED:1");
    wmanaged = ((npos > -1) ? 1 : 0);
}

```

```

if (utilities->stringSearch(wvalue, "TYPE:Wireless") >= 0)
    wtype = T_WIRELESS;
else if (utilities->stringSearch(wvalue, "TYPE:Wired") >= 0)
    wtype = T_WIRED;
else if (utilities->stringSearch(wvalue, "TYPE:Broadband(3G)") >= 0)
    wtype = T_BROADBAND;
else wtype = NA;

wdCreated = utilities->stripLabelT(wvalue, "DATECREATED:", '~');

wdConnected = utilities->stripLabelT(wvalue, "DATELASTCONNECTED:",

// Check if the network already exists
if (utilities->stringSearch("N/A", witem) == -1) {
    rc = sqlite3_bind_text(stmt_selNetwork, 1, witem, -1,
                           SQLITE_TRANSIENT);
    rc = sqlite3_bind_int(stmt_selNetwork, 2, wtype);
    rc = sqlite3_bind_int(stmt_selNetwork, 3, wmanaged);
    rc = sqlite3_step(stmt_selNetwork);
    if (rc == SQLITE_ROW) { // read row
        witemid = sqlite3_column_int(stmt_selNetwork, 0);
    }
    else {
        // If the network doesn't already exist, add it
        wsort = (char *)malloc(sizeof(char) * (strlen(witem) + 1));
        strcpy_s(wsort, strlen(witem) + 1, witem);
        witemid = insert_item(I_NETWORK, S_REGISTRY, wreg_id, wsort);
        printf("itemid = %i    witem = %s    wtype = %i    wmanaged = %i\n",
               witemid, witem, wtype, wmanaged);
        sqlite3_reset(stmt_insNetwork);
        rc = sqlite3_bind_int(stmt_insNetwork, 1, witemid); // ItemID
        printf("rc = %i\n", rc);
        rc = sqlite3_bind_text(stmt_insNetwork, 2, witem, -1, SQLITE_TRANSIENT);
        printf("rc = %i\n", rc);
        rc = sqlite3_bind_int(stmt_insNetwork, 3, wtype); // Type
        printf("rc = %i\n", rc);
        rc = sqlite3_bind_int(stmt_insNetwork, 4, wmanaged); // Managed
        printf("rc = %i\n", rc);
        rc = sqlite3_bind_int(stmt_insNetwork, 5, S_REGISTRY); // Trace
        printf("rc = %i\n", rc);
        rc = sqlite3_bind_int(stmt_insNetwork, 6, wreg_id); // Trace
        printf("rc = %i\n", rc);
        rc = sqlite3_step(stmt_insNetwork);
        printf("rc = %i\n", rc);
    }
}

```



```

        if (rc != SQLITE_DONE) { printf("Error: %i.  couldn't execute\n", rc);
    }
}
else {
    witemid = NA;
}

if (wdConnected > 0) {
    wactivity = insert_activity(NA, NA, A_CONNECT, witemid,
        S_REGISTRY, S_REGISTRY, wreg_id);
    insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
        wdConnected, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
}

if (wdCreated > 0) {
    wactivity = insert_activity(NA, NA, A_FCONNECT,
        witemid, S_REGISTRY, S_REGISTRY, wreg_id);
    insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
        wdCreated, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
}

free(witem);
}
else if ((utilities->stringSearch(wkey,
    "Microsoft/Windows/CurrentVersion/Explorer/WordWheelQuery") >
    || (utilities->stringSearch(wkey,
    "Microsoft/Windows/CurrentVersion/Explorer/RecentDocs") >= 0)
) {
    // Determine the action performed based on the key
    if (utilities->stringSearch(wkey,
        "Microsoft/Windows/CurrentVersion/Explorer/WordWheelQuery") >
        wactionid = A_SEARCH;
    else if (utilities->stringSearch(wkey,
        "Microsoft/Windows/CurrentVersion/Explorer/RecentDocs") >= 0)
        wactionid = A_OPEN;
    wseqid = utilities->stripN(wvalue, '~', 0);
    witem = utilities->strip(wvalue, '~', 1);

    // Check if the device already exists
    if (utilities->stringSearch("N/A", witem) == -1) {
        if (wactionid == A_SEARCH) {
            witemid = get_search(witem, P_EXPLORER, S_REGISTRY, wreg_id);
        }
    }
}

```



```

        free(witem);
    }
    else if (utilities->stringSearch(wkey,
        "Microsoft/Windows/CurrentVersion/Explorer/ComDlg32") > 0) {
        wseqid = utilities->stripN(wvalue, '~', 0);
        witem = utilities->strip(wvalue, '~', 1);

        // Determine the action performed based on the key
        if (utilities->stringSearch(wkey, "CIDSSizeMRU") >= 0) {
            witemid = get_application(witem, 0, S_REGISTRY, wreg_id);
            wactivity = insert_activity(wactor, P_EXPLORER,
                A_EXECUTE, witemid, S_REGISTRY, S_REGISTRY, wreg_id);
            insert_mru(wactivity, wseqid, n32time, S_REGISTRY,
                S_REGISTRY, wreg_id); \
        }
        else if ((utilities->stringSearch(wkey, "LastVisited") >= 0) ||
            (utilities->stringSearch(wkey, "FirstFolder") >= 0)) {
            i = 2;
            while (witem != NULL) {
                witemid = get_application(witem, 0, S_REGISTRY, wreg_id);
                wactivity = insert_activity(wactor, P_EXPLORER,
                    A_EXECUTE, witemid, S_REGISTRY, S_REGISTRY, wreg_id);
                insert_mru(wactivity, wseqid, n32time, S_REGISTRY,
                    S_REGISTRY, wreg_id); \
                witem = utilities->strip(wvalue, '~', i++);
            }
        }
        else if (utilities->stringSearch(wkey, "OpenSavePid") >= 0) {
            witemid = get_file(witem, S_REGISTRY, wreg_id);
            wactivity = insert_activity(wactor, P_EXPLORER, A_OPEN,
                witemid, S_REGISTRY, S_REGISTRY, wreg_id);
            insert_mru(wactivity, wseqid, n32time, S_REGISTRY,
                S_REGISTRY, wreg_id); \
        }

        free(witem);
    }
    else if (utilities->stringSearch(wkey,
        "Software/Microsoft/Windows/Shell/BagMRU") > 0) {
        wseqid = utilities->stripN(wvalue, '~', 0);
        witem = utilities->strip(wvalue, '~', 1);

```

```

    if (utilities->stringSearch("N/A", witem) == -1) {
        witemid = get_file(witem, S_REGISTRY, wreg_id);
    }
    else {
        witemid = NA;
    }

    wdAccessed = utilities->stripLabelT(wvalue, "LastAccessed:", '~');
    if (wdAccessed != 0) {
        wactivity = insert_activity(wactor, NA, A_ACCESS,
            witemid, S_REGISTRY, S_REGISTRY, wreg_id);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
            wdAccessed, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
    }
    else {
        wdAccessed = utilities->stripLabelT(wvalue, "LastModified:", '~');
        if (wdAccessed != 0) {
            wactivity = insert_activity(wactor, NA, A_ACCESS,
                witemid, S_REGISTRY, S_REGISTRY, wreg_id);
            insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                wdAccessed, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
        }
        else {
            wdAccessed = utilities->stripLabelT(wvalue, "Created:", '~');
            if (wdAccessed != 0) {
                wactivity = insert_activity(wactor, NA, A_ACCESS,
                    witemid, S_REGISTRY, S_REGISTRY, wreg_id);
                insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                    wdAccessed, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNb
                );
            }
            else {
                wactivity = insert_activity(wactor, NA, A_ACCESS,
                    witemid, S_REGISTRY, S_REGISTRY, wreg_id);
                insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
                    n32time, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
            }
        }
    }

    free(witem);
}

else if ((utilities->stringSearch(wkey, "Software/Microsoft/Office/14.
    (utilities->stringSearch(wkey, "File MRU") > 0)

```

```

    ) {
        wtemp = utilities->strip(wkey, '/', 5);
        if (utilities->stringSearchU(wtemp, "WORD") > -1)
            wappid = P_WORD;
        else if (utilities->stringSearchU(wtemp, "EXCEL") > -1)
            wappid = P_EXCEL;
        else if (utilities->stringSearchU(wtemp, "POWERPOINT") > -1)
            wappid = P_POWERPOINT;
        else if (utilities->stringSearchU(wtemp, "ACCESS") > -1)
            wappid = P_ACCESS;
        else
            wappid = P_OFFICE;
        free(wtemp);

        if (utilities->stringSearch("N/A", wvalue) == -1) {
            witemid = get_file(wvalue, S_REGISTRY, wreg_id);
        }
        else {
            witemid = NA;
        }

        wactivity = insert_activity(wactor, wappid, A_OPEN,
            witemid, S_REGISTRY, S_REGISTRY, wreg_id);
        insert_timeline(T_DATALOAD, wactivity, NA, R_AT,
            n32time, NA, S_REGISTRY, S_REGISTRY, wreg_id, caseNbr);
    }
else if (((utilities->stringSearch(wkey,
    "Software/Adobe/Acrobat Reader/") >= 0) &&
(utilities->stringSearch(wkey,
    "AVGeneral/cRecentFiles") > 0) &&
(utilities->stringSearch(wkey,
    "tDIText") > 0))
|| (utilities->stringSearch(wkey,
    "Software/Microsoft/Windows/Applets/Wordpad/Recent File List"
|| ((utilities->stringSearch(wkey,
    "Software/Microsoft/VCEXpress") >= 0) &&
(utilities->stringSearch(wkey, "FileMRUList") > 0))
) {
    if (parameters->debugParms >= 5) printf("here2b8\n");
    if ((utilities->stringSearch(wkey,
        "Software/Adobe/Acrobat Reader/") >= 0) &&
(utilities->stringSearch(wkey,
    "AVGeneral/cRecentFiles") > 0) &&

```

```

        (utilities->stringSearch(wkey, "tDIText") > 0))
        wappid = P_ADOBERDR;
    else if (utilities->stringSearch(wkey,
        "Software/Microsoft/Windows/Applets/Wordpad/Recent File List"
        wappid = P_WORDPAD;
    else if ((utilities->stringSearch(wkey,
        "Software/Microsoft/VCEXpress") >= 0) &&
        (utilities->stringSearch(wkey, "FileMRUList") > 0))
        wappid = P_VCEXPRESS;
    else
        wappid = NA;

    wseqid = utilities->stripN(wvalue, '~', 0);
    witem = utilities->strip(wvalue, '~', 1);
    if (utilities->stringSearch("N/A", witem) == -1) {
        witemid = get_file(witem, S_REGISTRY, wreg_id);
    }
    else {
        witemid = NA;
    }

    wactivity = insert_activity(wactor, wappid, A_OPEN,
        witemid, S_REGISTRY, S_REGISTRY, wreg_id);
    insert_mru(wactivity, wseqid, n32time, S_REGISTRY,
        S_REGISTRY, wreg_id); \

}
else if (utilities->stringSearch(wkey,
    "Software/Microsoft/Internet Explorer/TypedURLs/") >= 0) {
    if (parameters->debugParms >= 5) printf("here2b9\n");
    wseqid = utilities->stripN(wvalue, '~', 0);
    witem = utilities->strip(wvalue, '~', 1);
    if (utilities->stringSearch("N/A", witem) == -1) {
        witemid = get_url(witem, S_REGISTRY, wreg_id);
    }
    else {
        witemid = NA;
    }

    wactivity = insert_activity(wactor, P_IE, A_VISIT, witemid,
        S_REGISTRY, S_REGISTRY, wreg_id);
    insert_mru(wactivity, wseqid, n32time, S_REGISTRY, S_REGISTRY,
        wreg_id);

```

```

        }

        free(wtime); free(wkey); free(wvalue);

    }
    }
    fclose(dumpfile);
} while (FindNextFile(fh, fd));
}
FindClose(fh);

return 1;
} // processDir()

uint32_t BuildDB::last_insert(sqlite3 *db) {
    // Purpose: return the rowid of the last insert for the passed database

    sqlite3_stmt *stmt;
    char exec_string[1000];
    int rc;
    uint32_t wretval;

    //////////////////////////////////////

    sprintf_s(exec_string, 1000, "SELECT LAST_INSERT_ROWID()");
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 1000, &stmt, 0);
    if (rc != SQLITE_OK) {
        printf("Error: %i. couldn't prepare select statement\n", rc);
        return (1);
    }
    rc = sqlite3_step(stmt);
    if (rc == SQLITE_ROW) // read row
        wretval = sqlite3_column_int(stmt, 0);
    else
        wretval = NA;
    sqlite3_reset(stmt);
    sqlite3_finalize(stmt);

    return wretval;
}

```

```

////////////////////////////////////

```

```

//
// The following functions dump into the output files to be loaded later with import
//

uint32_t BuildDB::insert_event_support(uint32_t wactivityID, uint8_t winfo_type,
                                       char * winfo, uint8_t wtracetypeid, uint32_t wtraceid) {

    fprintf(eventSupport, "%i|i|s|s|i\n", wactivityID, winfo_type, winfo, " ", wtracetypeid, wtraceid);
    return 0;
}

uint32_t BuildDB::insert_activity(uint32_t wactorid, uint32_t wapplicationid, uint16_t wactionid,
                                  uint32_t witemid, uint8_t wsourcedid, uint8_t tracetypeid, uint32_t traceid) {

    fprintf(activities, "%i|i|i|i|i|i|i|i|i\n", nextActivity++, wactorid, wactionid, wapplicationid, witemid, wsourcedid, tracetypeid, traceid);
    return nextActivity - 1;
}

uint32_t BuildDB::insert_event(uint32_t woid, uint32_t n32time, uint32_t wappexec,
                               uint32_t wactor, uint32_t waction, char *wmessage) {

    fprintf(eventFile, "%i|i|i|i|i|i|i|i|i\n", nextEvent++, woid, n32time, wappexec, wactor, waction, wmessage);
    return nextEvent - 1;
}

uint32_t BuildDB::insert_timeline(uint8_t wtimelinetypeid, uint32_t wactivity1id,
                                  uint32_t wactivity2id, uint8_t wrelationshipid,
                                  uint32_t wtime1, uint32_t wtime2, uint8_t wsourcedid,
                                  uint8_t wtracetypeid, uint32_t wtraceid, TCHAR *caseNbr) {

    _ftprintf(timeline, L"%i|i|i|i|i|i|i|i|i|s|i|s\n", nextTimeline++, wtimelinetypeid, wactivity1id, wactivity2id, wrelationshipid, wtime1, wtime2, wsourcedid, L" ", wtracetypeid, wtraceid, caseNbr);
    return nextTimeline - 1;
}

```



```

uint32_t BuildDB::insert_mru(uint32_t wactivityid, uint32_t wseqid, uint32_t wtime,
                             uint8_t wsourceid, uint8_t wtracetypeid, uint32_t wtraceid) {

    fprintf(mruList, "%i|i|i|i|i|s|i\n",
            wactivityid, wseqid, wtime, wsourceid, " ", wtraceid);
    return 0;
}

uint32_t BuildDB::insert_registry(char * wregistryKey, char *wkeyType,
                                  char *wcharValue, uint32_t n32time) {

    for (int i = 0; i < strlen(wcharValue); i++) {
        if (wcharValue[i] == '|') wcharValue[i] = ' ';
    }

    fprintf(registry, "%i|s|s|s|0| |i\n", nextRegistry++, wregistryKey,
            wkeyType, wcharValue, n32time);
    return nextRegistry - 1;
}

////////////////////////////////////

//
// The following methods look to see if a master record already exists with the passe
// If it does, then it's returned; otherwise it's inserted directly into the new data
// There are some few records added to these tables, there's no good reason to delay
//
uint32_t BuildDB::get_appexec(char * wapplication, uint8_t wtracetypeid, uint32_t wtr

    int rc;
    uint32_t wappid;

    sqlite3_reset(stmt_selAppexec);
    rc = sqlite3_bind_text(stmt_selAppexec, 1, wapplication, -1, SQLITE_TRANSIENT);
    if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind Application\n", rc); return
    rc = sqlite3_step(stmt_selAppexec);
    if (rc == SQLITE_ROW) { // read row
        wappid = sqlite3_column_int(stmt_selAppexec, 0);
    }
    else {

```

```

        sqlite3_reset(stmt_insAppexec);
        rc = sqlite3_bind_text(stmt_insAppexec, 1, wapplication, -1, SQLITE_TRANSIENT);
        if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind Application\n", rc); re
        rc = sqlite3_bind_int(stmt_insAppexec, 2, wtracetypeid); // Trace Type
        if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind Control Type\n", rc); r
        rc = sqlite3_bind_int(stmt_insAppexec, 3, wtraceid); // Trace ID
        if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind Control ID\n", rc); ret
        rc = sqlite3_step(stmt_insAppexec);
        if (rc != SQLITE_DONE) { printf("Error: %i. couldn't execute insert appexecs\n
        if (rc == SQLITE_ROW) { // read row
            wappid = sqlite3_column_int(stmt_insAppexec, 0);
        }
        else
            wappid = NA;
    }

    return wappid;
}

uint32_t BuildDB::get_actor(TCHAR *wname) {

    int rc;

    char * sactor;
    uint32_t wactor, wlen, i;
    TCHAR *tpos, *tpos2;

    tpos = _tcschr(wname, '_');
    tpos2 = _tcsrchr(wname, '_');
    if (tpos2 != tpos) {
        wlen = (uint32_t)(_tcslen(tpos) - _tcslen(tpos2) - 1);
        if ((wlen >= 0) && (wlen < 0xff)) {
            sactor = (char *)malloc(sizeof(char) * (wlen + 1));
            // Convert TCHAR filename to CHAR actor -- doesn't handle all cases
            for (i = 0; i < wlen; i++) {
                sactor[i] = tpos[i + 1];
            }
            sactor[wlen] = 0;
            if (strlen(sactor) > 0) {
                sqlite3_reset(stmt_selActor);
                rc = sqlite3_bind_text(stmt_selActor, 1, sactor, -1, SQLITE_TRANSIENT);
                if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind registry key\n",
                rc = sqlite3_step(stmt_selActor);

```

```

        if (rc == SQLITE_ROW) { // read row
            wactor = sqlite3_column_int(stmt_selActor, 0);
        }
        else
            wactor = NA;
        // If the record hasn't been added yet
        if (wactor == NA) {
            sqlite3_reset(stmt_insActor);
            rc = sqlite3_bind_text(stmt_insActor, 1, sactor, -1, SQLITE_TRANSIENT);
            if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind Actor Name\n", rc);
            rc = sqlite3_step(stmt_insActor);
            if (rc != SQLITE_DONE) { printf("Error: %i. couldn't execute\n", rc);
            // Get the automatic ID created for activity
            wactor = last_insert(parameters->db);
        }
    }
    else
        wactor = NA;
    free(sactor);
}
else
    wactor = NA;
}
else
    wactor = NA;

return wactor;
}

uint32_t BuildDB::get_actor2(char *wuser) {

    int rc;

    uint32_t wactor;

    if (wuser == NULL) return NA;
    if (strlen(wuser) > 0) {
        sqlite3_reset(stmt_selActor2);
        rc = sqlite3_bind_text(stmt_selActor2, 1, wuser, -1, SQLITE_TRANSIENT);
        if (rc != SQLITE_OK) { printf("Error: %i. couldn't bind actor name\n", rc); ret
        rc = sqlite3_step(stmt_selActor2);
        if (rc == SQLITE_ROW) { // read row
            wactor = sqlite3_column_int(stmt_selActor2, 0);

```



```

        wlinfo->traceType = wtracetypeid;
        g_hash_table_insert(htApplications, wlinfo->applicationName, wlinfo);
    }

    return witemid;
}

uint32_t BuildDB::get_file(char * wfilename, uint8_t wtracetypeid, uint32_t wtraceid)

    uint32_t witemid, whash, i;
    file_t *wlinfo;

    wlinfo = (file_t *)g_hash_table_lookup(htFiles, wfilename);
    if (wlinfo != NULL)
        witemid = wlinfo->itemid;
    else {
        wlinfo = (file_t *)malloc(sizeof(file_t) * 1);
        wlinfo->fileName = (char *)malloc(sizeof(char) * (strlen(wfilename) + 1));
        for (i = 0; i < strlen(wfilename); i++) wlinfo->fileName[i] = wfilename[i]; wlinfo->fileName[i] = '\0';
        witemid = insert_item(I_FILE, wtracetypeid, wtraceid, wlinfo->fileName);
        wlinfo->itemid = witemid;
        whash = 0;
        for (i = 0; i < strlen(wfilename); i++) whash += wfilename[i];
        wlinfo->hash = whash;
        wlinfo->traceid = wtraceid;
        wlinfo->traceType = wtracetypeid;
        g_hash_table_insert(htFiles, wlinfo->fileName, wlinfo);
    }

    return witemid;
}

uint32_t BuildDB::get_url(char * wurl, uint8_t wtracetypeid, uint32_t wtraceid) {

    uint32_t witemid, i;
    url_t *wlinfo;

    wlinfo = (url_t *)g_hash_table_lookup(htUrls, wurl);
    if (wlinfo != NULL)
        witemid = wlinfo->itemid;
    else {
        wlinfo = (url_t *)malloc(sizeof(url_t) * 1);
        wlinfo->urlName = (char *)malloc(sizeof(char) * (strlen(wurl) + 2));

```

```

        for (i = 0; i < strlen(wurl); i++) wlinfo->urlName[i] = wurl[i]; wlinfo->urlName[
        witemid = insert_item(I_URL, wtracetypeid, wtraceid, wlinfo->urlName);
        wlinfo->itemid = witemid;
        wlinfo->traceid = wtraceid;
        wlinfo->traceType = wtracetypeid;
        g_hash_table_insert(htUrls, (wlinfo->urlName), wlinfo);
    }

    return witemid;
}

uint32_t BuildDB::get_key(char * wkey, uint8_t wtracetypeid, uint32_t wtraceid) {

    uint32_t witemid, i;
    key_t *wlinfo;

    wlinfo = (key_t *)g_hash_table_lookup(htKeys, wkey);
    if (wlinfo != NULL)
        witemid = wlinfo->itemid;
    else {
        wlinfo = (key_t *)malloc(sizeof(key_t) * 1);
        wlinfo->keyName = (char *)malloc(sizeof(char) * (strlen(wkey) + 2));
        for (i = 0; i < strlen(wkey); i++) wlinfo->keyName[i] = wkey[i]; wlinfo->keyName[
        witemid = insert_item(I_KEY, wtracetypeid, wtraceid, wlinfo->keyName);
        wlinfo->itemid = witemid;
        wlinfo->traceid = wtraceid;
        wlinfo->traceType = wtracetypeid;
        g_hash_table_insert(htKeys, (wlinfo->keyName), wlinfo);
    }

    return witemid;
}

uint32_t BuildDB::get_search(char * wsearchkey, uint8_t wapplication, uint8_t wtracet

    uint32_t i, witemid;
    char *wstring;
    search_t *wlinfo;

    wstring = (char *)malloc(sizeof(char) * (strlen(wsearchkey) + 1 + 2));
    sprintf_s(wstring, strlen(wsearchkey) + 2, "%s|%1i", wsearchkey, wapplication);
    wlinfo = (search_t *)g_hash_table_lookup(htSearches, wstring);
    if (wlinfo != NULL)

```

```

        witemid = wlinfo->itemid;
    else {
        wlinfo = (search_t *)malloc(sizeof(search_t) * 1);
        wlinfo->searchKey = (char *)malloc(sizeof(char) * (strlen(wsearchkey) + 1));
        for (i = 0; i < strlen(wsearchkey); i++) wlinfo->searchKey[i] = wsearchkey[i];
        witemid = insert_item(I_SEARCH, wtracetypeid, wtraceid, wlinfo->searchKey);
        wlinfo->application = wapplication;
        wlinfo->itemid = witemid;
        wlinfo->traceid = wtraceid;
        wlinfo->traceType = wtracetypeid;
        g_hash_table_insert(htSearches, wstring, wlinfo);
    }

    return witemid;
}

uint32_t BuildDB::insert_item(uint8_t witemtypeid, uint8_t tracetypeid, uint32_t traceid,
// Purpose: add the item into the item hash table for later dumping
    item_t *wlinfo;
    uint32_t *witem;

    witem = (uint32_t *)malloc(sizeof(uint32_t) * 1);
    wlinfo = (item_t *)malloc(sizeof(item_t) * 1);

    wlinfo->itemid = nextItem++;
    *witem = wlinfo->itemid;
    wlinfo->itemtypeid = witemtypeid;
    wlinfo->traceid = traceid;
    wlinfo->traceType = tracetypeid;
    wlinfo->sort = wsort;
    g_hash_table_insert(htItems, witem, wlinfo);

    return wlinfo->itemid;
}

```

8.4 ItemUtilities

Header.

```
#pragma once
#include "TearCPP.h"
#include "Parameters.h"
#include "Generics.h"
class Parameters;

#define htItem_i std::unordered_map<uint32_t, ItemUtilities::item_t *>::const_iterator

class ItemUtilities
{
    Parameters *parameters;
    Generics *generics;

    sqlite3_stmt *stmt_devices;
    sqlite3_stmt *stmt_networks;
    sqlite3_stmt *stmt_searches;
    sqlite3_stmt *stmt_files;
    sqlite3_stmt *stmt_applications;
    sqlite3_stmt *stmt_urls;
    sqlite3_stmt *stmt_keys;
    sqlite3_stmt *stmt_services;
    sqlite3_stmt *stmt_downloads;
    sqlite3_stmt *stmt_selItem;
    sqlite3_stmt *stmt_patternGenerics;

public:
    typedef struct item_type {
        uint32_t itemid;
        char *itemName;
        uint32_t itemtype;
    } item_t;

    std::unordered_map<uint32_t, item_t *> *htItems;

    ~ItemUtilities() {}

    ItemUtilities(Parameters *wparameters, Generics *wgenerics);
    char * getItemName(Generics *generics, uint32_t witemid);
    uint32_t getItemtype(uint32_t witemid);
```



```

        bool sameAs(uint32_t witem1, uint32_t witem2);

};

C++.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// Class: ItemUtilities                                                                                   //
// Notes:                                                                                               //
//      1. This class contains utilities for processing items. It is                               //
//          separated from the utilities class to avoid circular class creations//
// Initial Release: 29-Sep-17                                                                                   //
//                                                                                               //
// Revision History:                                                                                   //
//                                                                                               //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#pragma once
#include "ItemUtilities.h"

ItemUtilities::ItemUtilities(Parameters *wparameters, Generics *wgenerics) {

    sqlite3_stmt *stmt;
    char exec_string[2000];
    int rc;
    item_t *itemDef;

    parameters = wparameters;
    generics = wgenerics;

    // For GetItemName
    sprintf_s(exec_string, 2000, "SELECT DeviceID FROM devices where devices.itemid = %d", witem1);
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_devices), 0);

    sprintf_s(exec_string, 2000, "SELECT NetworkId FROM networks where networks.itemid = %d", witem1);
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_networks), 0);

    sprintf_s(exec_string, 2000, "SELECT SearchKey FROM searches where searches.itemid = %d", witem1);
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_searches), 0);

    sprintf_s(exec_string, 2000, "SELECT Filename FROM files where files.itemid = %d", witem1);
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_files), 0);

```

```

sprintf_s(exec_string, 2000, "SELECT ApplicationName FROM applications where
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_application

sprintf_s(exec_string, 2000, "SELECT UrlName FROM urls where urls.itemid = ?1
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_urls), 0);

sprintf_s(exec_string, 2000, "SELECT KeyName FROM keys where keys.itemid = ?1
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_keys), 0);

sprintf_s(exec_string, 2000, "SELECT ServiceID FROM services where services.i
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_services),

sprintf_s(exec_string, 2000, "SELECT UrlId FROM downloads where downloads.act
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_downloads),

sprintf_s(exec_string, 2000, "SELECT pattern from patterns,patterngenerics wh
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_patternGene

// Used in GetItemName and GetItemType
sprintf_s(exec_string, 2000, "SELECT itemtypeid from items where itemid = ?1
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &(stmt_selItem), 0

// The following loads the items and patterns into has tables for faster proc
// However, there is an upfront performance hit, so the user is given the opt
// this or to use the parameters->db each time
htItems = new          std::unordered_map<uint32_t, item_t *>;
if (parameters->useItemPatternLists) {
    printf("Loading Items\n");
    sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, filena
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        itemDef = (item_t *)malloc(sizeof(item_t) * 1);
        itemDef->itemid = sqlite3_column_int(stmt, 0);
        itemDef->itemtype = sqlite3_column_int(stmt, 1);
        itemDef->itemName = (char *)malloc(sizeof(char) *
                                (strlen((char *)sqlite3_column_text(s
                                strlen((char *)
                                sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
        (*htItems)[itemDef->itemid] = itemDef;
    }
}

```

```

sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, keynam
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
    itemDef = (item_t *)malloc(sizeof(item_t) * 1);
    itemDef->itemid = sqlite3_column_int(stmt, 0);
    itemDef->itemtype = sqlite3_column_int(stmt, 1);
    itemDef->itemName = (char *)malloc(sizeof(char) *
        (strlen((char *)sqlite3_column_text(s
strcpy_s(itemDef->itemName, strlen((char *)
        sqlite3_column_text(stmt, 2)) + 1, (char *)sq
    (*htItems)[itemDef->itemid] = itemDef;
}

```

```

sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, urlnam
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
    itemDef = (item_t *)malloc(sizeof(item_t) * 1);
    itemDef->itemid = sqlite3_column_int(stmt, 0);
    itemDef->itemtype = sqlite3_column_int(stmt, 1);
    itemDef->itemName = (char *)malloc(sizeof(char) *
        (strlen((char *)sqlite3_column_text(s
strcpy_s(itemDef->itemName, strlen((char *)
        sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
    (*htItems)[itemDef->itemid] = itemDef;
}

```

```

sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, applic
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
    itemDef = (item_t *)malloc(sizeof(item_t) * 1);
    itemDef->itemid = sqlite3_column_int(stmt, 0);
    itemDef->itemtype = sqlite3_column_int(stmt, 1);
    itemDef->itemName = (char *)malloc(sizeof(char) *
        (strlen((char *)sqlite3_column_text(s
strcpy_s(itemDef->itemName, strlen((char *)
        sqlite3_column_text(stmt, 2)) + 1, (char *)sq
    (*htItems)[itemDef->itemid] = itemDef;
}

```

```

sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, device
rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
    itemDef = (item_t *)malloc(sizeof(item_t) * 1);

```

```

        itemDef->itemid = sqlite3_column_int(stmt, 0);
        itemDef->itemtype = sqlite3_column_int(stmt, 1);
        itemDef->itemName = (char *)malloc(sizeof(char) *
            (strlen((char *)sqlite3_column_text(stmt, 2)) + 2));
        strcpy_s(itemDef->itemName, strlen((char *)
            sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
            (*htItems)[itemDef->itemid] = itemDef;
    }

```

```

    sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, servic
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        itemDef = (item_t *)malloc(sizeof(item_t) * 1);
        itemDef->itemid = sqlite3_column_int(stmt, 0);
        itemDef->itemtype = sqlite3_column_int(stmt, 1);
        itemDef->itemName = (char *)malloc(sizeof(char) *
            (strlen((char *)sqlite3_column_text(stmt, 2)) + 2));
        strcpy_s(itemDef->itemName, strlen((char *)
            sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
            (*htItems)[itemDef->itemid] = itemDef;
    }

```

```

    sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, network
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        itemDef = (item_t *)malloc(sizeof(item_t) * 1);
        itemDef->itemid = sqlite3_column_int(stmt, 0);
        itemDef->itemtype = sqlite3_column_int(stmt, 1);
        itemDef->itemName = (char *)malloc(sizeof(char) *
            (strlen((char *)sqlite3_column_text(stmt, 2)) + 2));
        strcpy_s(itemDef->itemName, strlen((char *)
            sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
            (*htItems)[itemDef->itemid] = itemDef;
    }

```

```

    sprintf_s(exec_string, 2000, "SELECT items.itemid, itemtypeid, search
    rc = sqlite3_prepare_v2(parameters->db, exec_string, 2000, &stmt, 0);
    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        itemDef = (item_t *)malloc(sizeof(item_t) * 1);
        itemDef->itemid = sqlite3_column_int(stmt, 0);
        itemDef->itemtype = sqlite3_column_int(stmt, 1);
        itemDef->itemName = (char *)malloc(sizeof(char) *
            (strlen((char *)sqlite3_column_text(stmt, 2)) + 2));
    }

```

```

        strcpy_s(itemDef->itemName, strlen((char *)
            sqlite3_column_text(stmt, 2)) + 1, (char *)sqlite3_co
            (*htItems)[itemDef->itemid] = itemDef;
    }
}

}

////////////////////////////////////
// Retrieve the item name //
////////////////////////////////////
char * ItemUtilities::getItemName(Generics *generics, uint32_t witemid) {
    htItem_i htiter;
    item_t *itemDef;
    int rc2;
    char *witem_c;
    uint32_t witemtypeid, wgenericName;

    witem_c = (char *)malloc(sizeof(char) * 255);
    if (witem_c == NULL) return NULL;

    htiter = htItems->find(witemid);
    if (htiter != htItems->end()) {
        itemDef = htiter->second;
        if (itemDef->itemName != NULL) {
            strncpy_s(witem_c, 251, itemDef->itemName, min(250, strlen(it
            return witem_c;
        }
    }

    sqlite3_reset(stmt_selItem);
    sqlite3_bind_int(stmt_selItem, 1, witemid);
    sqlite3_step(stmt_selItem);
    witemtypeid = sqlite3_column_int(stmt_selItem, 0);

    // Catch whatever we missed
    if (witemtypeid == NA) {
        witem_c[0] = 'N'; witem_c[1] = '/'; witem_c[2] = 'A'; witem_c[3] = 0;
    }
    else if (witemtypeid == I_GENERIC) {
        sqlite3_reset(stmt_patternGenerics);

```

```

rc2 = sqlite3_bind_int(stmt_patternGenerics, 1, witemid);
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't bind Item ID\n", rc2);
    return NULL;
}
rc2 = sqlite3_step(stmt_patternGenerics);
wgenericName = generics->getGeneric(witemid);
if (rc2 == SQLITE_ROW)
    sprintf_s(witem_c, 249, "Generic Item %u (%s)", wgenericName,
        (const char *)sqlite3_column_text(stmt_patternGenerics, 0));
else
    sprintf_s(witem_c, 249, "Generic Item %u", wgenericName);
witem_c[249] = 0;
}
else if (witemtypeid == I_FILE) {
    sqlite3_reset(stmt_files);
    rc2 = sqlite3_bind_int(stmt_files, 1, witemid);
    if (rc2 != SQLITE_OK) {
        printf("Error: %i. couldn't bind Item ID\n", rc2);
        return NULL;
    }
    rc2 = sqlite3_step(stmt_files);
    if (rc2 == SQLITE_ROW)
        strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_files, 0));
    else
        sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
    witem_c[249] = 0;
}
else if (witemtypeid == I_DEVICE) {
    sqlite3_reset(stmt_devices);
    rc2 = sqlite3_bind_int(stmt_devices, 1, witemid);
    if (rc2 != SQLITE_OK) {
        printf("Error: %i. couldn't bind Device ID\n", rc2);
        return NULL;
    }
    rc2 = sqlite3_step(stmt_devices);
    if (rc2 == SQLITE_ROW)
        strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_devices, 0));
    else
        sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
    witem_c[249] = 0;
}
else if (witemtypeid == I_NETWORK) {

```

```

sqlite3_reset(stmt_networks);
rc2 = sqlite3_bind_int(stmt_networks, 1, witemid);
if (rc2 != SQLITE_OK) {
    printf("Error: %i. couldn't bind Network ID\n", rc2);
    return NULL;
}
rc2 = sqlite3_step(stmt_networks);
if (rc2 == SQLITE_ROW)
    strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_networks, 0), 250);
else
    sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
witem_c[249] = 0;
}
else if (witemtypeid == I_SEARCH) {
    sqlite3_reset(stmt_searches);
    rc2 = sqlite3_bind_int(stmt_searches, 1, witemid);
    if (rc2 != SQLITE_OK) {
        printf("Error: %i. couldn't bind Search ID\n", rc2);
        return NULL;
    }
    rc2 = sqlite3_step(stmt_searches);
    if (rc2 == SQLITE_ROW)
        strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_searches, 0), 250);
    else
        sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
    witem_c[249] = 0;
}
else if (witemtypeid == I_APPLICATION) {
    sqlite3_reset(stmt_applications);
    rc2 = sqlite3_bind_int(stmt_applications, 1, witemid);
    if (rc2 != SQLITE_OK) {
        printf("Error: %i. couldn't bind Application ID\n", rc2);
        return NULL;
    }
    rc2 = sqlite3_step(stmt_applications);
    if (rc2 == SQLITE_ROW)
        strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_applications, 0), 250);
    else
        sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
    witem_c[249] = 0;
}
else if (witemtypeid == I_URL) {
    sqlite3_reset(stmt_urls);

```

```

        rc2 = sqlite3_bind_int(stmt_urls, 1, witemid);
        if (rc2 != SQLITE_OK) {
            printf("Error: %i. couldn't bind URL ID\n", rc2);
            return NULL;
        }
        rc2 = sqlite3_step(stmt_urls);
        if (rc2 == SQLITE_ROW)
            strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_urls, 0), 250);
        else
            sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
        witem_c[249] = 0;
    }
    else if (witemtypeid == I_KEY) {
        sqlite3_reset(stmt_keys);
        rc2 = sqlite3_bind_int(stmt_keys, 1, witemid);
        if (rc2 != SQLITE_OK) {
            printf("Error: %i. couldn't bind Key ID\n", rc2);
            return NULL;
        }
        rc2 = sqlite3_step(stmt_keys);
        if (rc2 == SQLITE_ROW)
            strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_keys, 0), 250);
        else
            sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
        witem_c[249] = 0;
    }
    else if (witemtypeid == I_SERVICE) {
        sqlite3_reset(stmt_services);
        rc2 = sqlite3_bind_int(stmt_services, 1, witemid);
        if (rc2 != SQLITE_OK) {
            printf("Error: %i. couldn't bind Service ID\n", rc2);
            return NULL;
        }
        rc2 = sqlite3_step(stmt_services);
        if (rc2 == SQLITE_ROW)
            strncpy_s(witem_c, 251, (const char *)sqlite3_column_text(stmt_services, 0), 250);
        else
            sprintf_s(witem_c, 249, "Non-Generic Item %u", witemid);
        witem_c[249] = 0;
    }
    else {
        witem_c[0] = 'N'; witem_c[1] = '/'; witem_c[2] = 'A'; witem_c[3] = 0;
    }
}

```



```

        sqlite3_reset(stmt_downloads);
        sqlite3_reset(stmt_devices);
        sqlite3_reset(stmt_files);
        sqlite3_reset(stmt_applications);
        sqlite3_reset(stmt_searches);
        sqlite3_reset(stmt_networks);
        sqlite3_reset(stmt_urls);
        sqlite3_reset(stmt_keys);

        return witem_c;
}

////////////////////////////////////
// Retrieve the item type //
////////////////////////////////////
uint32_t ItemUtilities::getItemtype(uint32_t witemid) {
    htItem_i htiter;
    item_t *itemDef;

    // only speeds up file names and keys, but that's most of them
    htiter = htItems->find(witemid);
    if (htiter != htItems->end()) {
        itemDef = htiter->second;
        return itemDef->itemtype;
    }

    sqlite3_reset(stmt_selItem);
    sqlite3_bind_int(stmt_selItem, 1, witemid);
    sqlite3_step(stmt_selItem);
    return sqlite3_column_int(stmt_selItem, 0);
}

////////////////////////////////////
// Checks if the two items are the same //
// -- for this, one having ($FILE_NAME) doesn't //
// affect whether they're considered equal //
////////////////////////////////////
bool ItemUtilities::sameAs(uint32_t witem1, uint32_t witem2) {
    htItem_i htiter;

```

```

uint32_t witemtype1, witemtype2;
char *name1, *name2, *start1, *start2, *test1, *test2;
int i;
bool free1, free2, result;

htiter = htItems->find(witem1);
if (htiter == htItems->end()) return (witem1 == witem2);
witemtype1 = htiter->second->itemtype;
if (witemtype1 != I_FILE) return (witem1 == witem2);
name1 = htiter->second->itemName;

htiter = htItems->find(witem2);
if (htiter == htItems->end()) return (witem1 == witem2);
witemtype2 = htiter->second->itemtype;
if (witemtype2 != I_FILE) return (witem1 == witem2);
name2 = htiter->second->itemName;

free1 = false;
free2 = false;

start1 = strstr(name1, " ($FILE_NAME)");
start2 = strstr(name2, " ($FILE_NAME)");
if (start1 == NULL && start2 == NULL) return (witem1 == witem2);
if (start1 != NULL) {
    free1 = true;
    test1 = (char *)malloc(sizeof(char) * (strlen(name1) - 13 + 2));
    int j = 0;
    for (i = 0; i < strlen(name1); i++) {
        if ((name1 + i < start1) || (name1 + i > start1 + 12))
            test1[j++] = name1[i];
    }
    test1[j] = 0;
}
else
    test1 = name1;

if (start2 != NULL) {
    free2 = true;
    test2 = (char *)malloc(sizeof(char) * (strlen(name2) - 13 + 2));
    int j = 0;
    for (i = 0; i < strlen(name2); i++) {
        if ((name2 + i < start2) || (name2 + i > start2 + 12))

```

```

                                test2[j++] = name2[i];
                                }
                                test2[j] = 0;
                                }
                                else
                                test2 = name2;

                                result = (strcmp(test1, test2) == 0);
                                if (free1) free(test1);
                                if (free2) free(test2);
                                return result;
}

```

8.5 Utilities

Header.

```

#pragma once
#include "TearCPP.h"

class Utilities
{
    uint8_t lastChar;
    uint8_t isDaylightSavingsTime;
public:
    Utilities() {
        isDaylightSavingsTime = 1;
        lastChar = 0;
    }
    ~Utilities() {}

    //////////////////////////////////////
    // Checks if the passed char* is a valid number //
    //////////////////////////////////////
    int isnum(char *wstr) {
        int i;
        for (i = 0; i < strlen(wstr); i++) {
            if ((i > 1) && (wstr[i] == ' ')) break;
            if ((wstr[i] > '9') || (wstr[i] < '0')) { return 0; }
        }
        return 1;
    }
}

```

```
}
```

```

////////////////////////////////////
// Return a malloc char* of N/A //
////////////////////////////////////
char * na_char() {
    char *wretstr;

    wretstr = (char *)malloc(sizeof(char) * 4);
    wretstr[0] = (char) 'N'; wretstr[1] = (char) 'A';
    wretstr[2] = (char) 'A'; wretstr[3] = (char) 0;

    return wretstr;
}

```

```

////////////////////////////////////
// strcmpU //
// //
// Compares two strings and finds the first place they differ but is //
// case insensitive //
// //
// Receives: //
//     two strings //
// Returns: //
//     <0 if the first character that doesn't match has a lower //
//         value in the first string than the second string //
//     >0 if the first character that doesn't match has a greater //
//         value in the first string than the second string //
//     0  the content of both strings is equal //
////////////////////////////////////

```

```

int strcmpU(char *str1, char *str2) {
    uint32_t i;
    uint8_t ch1, ch2;

    if ((str1 == NULL) && (str2 != NULL)) { return -1; }
    if ((str2 == NULL) && (str1 != NULL)) { return 1; }
    if ((str1 == NULL) || (str2 == NULL)) return -1;
    if ((strlen(str1) == 0) || (strlen(str2) == 0)) return -1;

    for (i = 0; i <= strlen(str1); i++) {

```

```

        ch1 = toupper(str1[i]);
        ch2 = toupper(str2[i]);
        if (ch1 < ch2) { return (-1 * (i + 1)); }
        if (ch1 > ch2) { return (i + 1); }
        if (ch1 == 0) { return 0; }
    }

    return 0;
}

/***** CONVERTING STRING TO UPPERCASE *****/

////////////////////////////////////
// Convert the string to upper case in place (i.e., do not return a new string) //
////////////////////////////////////
void strtoupperIP(char *lower) {
    unsigned int i;
    if (lower == NULL) return;
    for (i = 0; i < strlen(lower); i++)    lower[i] = toupper(lower[i]);
}

////////////////////////////////////
// Convert the string to upper case //
////////////////////////////////////
TCHAR * _tstrtoupper(TCHAR *lower) {
    unsigned int i;
    TCHAR *upper;
    upper = (TCHAR *)malloc(sizeof(TCHAR) * (_tcslen(lower) + 1));
    for (i = 0; i < _tcslen(lower); i++)    upper[i] = _totupper(lower[i]);
    upper[_tcslen(lower)] = 0;
    return upper;
}

////////////////////////////////////
// Convert the string to upper case //
////////////////////////////////////
CHAR * strtoupper(char *lower) {
    unsigned int i;
    CHAR *upper;
    upper = (CHAR *)malloc(sizeof(CHAR) * (strlen(lower) + 1));
    for (i = 0; i < strlen(lower); i++)    upper[i] = toupper(lower[i]);
    upper[strlen(lower)] = 0;
    return upper;
}

```

```

}

/***** SHOW PROGRESS *****/

////////////////////////////////////
// Increment the counter and if it's time, show progress //
////////////////////////////////////
uint8_t showTime(uint8_t crlf, uint32_t *ctr, uint32_t *lastseen, uint32_t incr) {
    time_t wtimer;
    struct tm timeinfo;
    char showtime[80];

    if (ctr != NULL && lastseen != NULL) {
        (*ctr)++;
        if (*ctr > *lastseen) (*lastseen) += incr; else return 0;
    }
    time(&wtimer);
    localtime_s(&timeinfo, &wtimer);
    strftime(showtime, 80, "Current time: %X", &timeinfo);
    if (ctr != NULL) printf("Processed %u \t\t%s", *ctr, showtime);
    else printf("%s", showtime);
    if (crlf) printf("\n");
    return 1;
}

void showClock() { printf("Clock time = %i\n", (uint32_t)clock()); }

/***** MATCH REGULAR EXPRESSION *****/

/* matchstar: search for c*regexp at beginning of text */
int matchstar(int c, char *regexp, char *text)
{
    do { /* a * matches zero or more instances */
        if (matchhere(regexp, text))
            return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text)

```

```

{
    if (regexp[0] == '\\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp + 2, text);
    if (regexp[0] == '$' && regexp[1] == '\\0')
        return *text == '\\0';
    if (*text != '\\0' && (regexp[0] == '.' || regexp[0] == *text))
        return matchhere(regexp + 1, text + 1);
    return 0;
}

/* match: search for regexp anywhere in text */
int match(char *regexp, char *text)
{
    if ((regexp == NULL) || (text == NULL)) return 1;

    if (regexp[0] == '^')
        return matchhere(regexp + 1, text);
    do { /* must look even if string is empty */
        if (matchhere(regexp, text))
            return 1;
    } while (*text++ != '\\0');
    return 0;
}

int matchu(char *regexp, char *text) {
    char *regexpU, *textU;
    int i, retval;

    regexpU = (char *)malloc(sizeof(char) * (strlen(regexp) + 1));
    if (regexpU == NULL) return 0;
    textU = (char *)malloc(sizeof(char) * (strlen(text) + 1));
    if (textU == NULL) return 0;
    for (i = 0; i < strlen(regexp); i++)
        regexpU[i] = toupper(regexp[i]); regexpU[i] = 0;
    for (i = 0; i < strlen(text); i++)
        textU[i] = toupper(text[i]); textU[i] = 0;
    retval = match(regexpU, textU);
    free(regexpU); free(textU);
    return retval;
}

```

```

/*****LIKE STRSTR but allows # or char specifiers and wildcards
-- less powerful than match*****/

/* searchString - return position of String2 in String1 */
int searchString(char wstring[], char wsubstr[]) {
    return searchStringN(wstring, wsubstr, 0);
}

/* searchString - return position of String2 in String1 */
int searchStringN(char wstring[], char wsubstr[], uint16_t nstart) {
    int i, j, firstOcc;
    i = nstart;
    j = 0;

    if (nstart > strlen(wstring)) return -1;

    if ((wstring == NULL) || (wsubstr == NULL)) return -1;
    if ((strlen(wstring) == 0) || (strlen(wsubstr) == 0)) return -1;
    while (wstring[i] != 0) {
        if (i >= strlen(wstring)) return -1;
        if (wsubstr[0] != '.') {
            while ((i < strlen(wstring)) && (wstring[i] != 0) &&
                !((wstring[i] == wsubstr[0]) ||
                ((wstring[i] >= '0') && (wstring[i] <= '9') &&
                (wsubstr[0] == '#')) ||
                ((wstring[i] >= 'a') && (wstring[i] <= 'z') &&
                (wsubstr[0] == '@')) ||
                ((wstring[i] >= 'A') && (wstring[i] <= 'Z') &&
                (wsubstr[0] == '@'))
            )
                i++;
        }
        if (i >= strlen(wstring)) return -1;
        if (wstring[i] == 0) return(-1); //search can not continue

        firstOcc = i;

        while (((wstring[i] == wsubstr[j]) || (wsubstr[j] == '.')) ||
            ((wsubstr[j] == '#') && (wstring[i] >= '0') && (wstring[i] <= '9')) ||
            ((wsubstr[j] == '@') && (((wstring[i] >= 'a') && (wstring[i] <= 'z')) ||
            ((wstring[i] >= 'A') && (wstring[i] <= 'Z')))))

```



```

        ) && (wstring[i] != 0) && (wsubstr[j] != 0) &&
        (i < strlen(wstring)) && (j < strlen(wsubstr))
    ) {
        i++;
        j++;
    }

    if (wsubstr[j] == 0) {
        return(firstOcc);
    }
    if (wstring[i] == 0) return(-1);

    i = firstOcc + 1;
    j = 0;
}
return -1;
}

/* stringSearch - return position of String2 in String1 */
int stringSearchU(char wstring[], char wsubstr[]) {
    return stringSearchUN(wstring, wsubstr, 0);
}

/* stringSearch - return position of String2 in String1 */
int stringSearchUN(char wstring[], char wsubstr[], uint16_t nstart) {
    int i, j, firstOcc;
    uint8_t wdebug;
    i = nstart;
    j = 0;

    if (nstart > strlen(wstring)) return -1;

    wdebug = (wsubstr[0] == '#') && (wstring[0] >= '0') && (wstring[0] <= '9');
    wdebug = 0;
    while (wstring[i] != 0) {
        if (i >= strlen(wstring)) return -1;
        if (wsubstr[0] != '.') {
            while ((i < strlen(wstring)) && (wstring[i] != 0) &&
                !((toupper(wstring[i]) == toupper(wsubstr[0])) ||
                ((wstring[i] >= '0') && (wstring[i] <= '9') && (wsubstr[0] == '#')) ||
                ((wstring[i] >= 'a') && (wstring[i] <= 'z') && (wsubstr[0] == '@'))
                ((wstring[i] >= 'A') && (wstring[i] <= 'Z') && (wsubstr[0] == '@'))
                )
            )

```

```

        )
        i++;
    }
    if (i >= strlen(wstring)) return -1;
    if (wstring[i] == 0) return(-1);        //search can not continue

    firstOcc = i;

    if (wdebug) printf("string[%i] = %c\n", i, wstring[i]);
    while (((toupper(wstring[i]) == toupper(wsubstr[j])) || (wsubstr[j] == '.'))
        ((wsubstr[j] == '#') && (wstring[i] >= '0') && (wstring[i] <= '9')) ||
        ((wsubstr[j] == '@') && ((wstring[i] >= 'a') && (wstring[i] <= 'z')) ||
        ((wstring[i] >= 'A') && (wstring[i] <= 'Z'))))
        ) &&
        (wstring[i] != 0) && (wsubstr[j] != 0) &&
        (i < strlen(wstring)) && (j < strlen(wsubstr))
    ) {
        if (wdebug)
            printf("\tstring[%i] (%c) = substr[%i] (%c)\n",
                i, wstring[i], j, wsubstr[j]);

        i++;
        j++;
    }

    if (wsubstr[j] == 0) {
        return(firstOcc);
    }
    if (wstring[i] == 0) return(-1);

    i = firstOcc + 1;
    j = 0;
}
return -1;
}

```

/***** HASH ROUTINES *****/

```

# define HASH_LITTLE_ENDIAN 1

#define hashsize(n) ((uint32_t)1<<(n))
#define hashmask(n) (hashsize(n)-1)
#define rot(x,k) (((x)<<(k)) | ((x)>>(32-(k))))

```

```

/*
-----
mix -- mix 3 32-bit values reversibly.

This is reversible, so any information in (a,b,c) before mix() is
still in (a,b,c) after mix().

If four pairs of (a,b,c) inputs are run through mix(), or through
mix() in reverse, there are at least 32 bits of the output that
are sometimes the same for one pair and different for another pair.
This was tested for:
* pairs that differed by one bit, by two bits, in any combination
of top bits of (a,b,c), or in any combination of bottom bits of
(a,b,c).
* "differ" is defined as +, -, ^, or ~^. For + and -, I transformed
the output delta to a Gray code (a^(a>>1)) so a string of 1's (as
is commonly produced by subtraction) look like a single 1-bit
difference.
* the base values were pseudorandom, all zero but one bit set, or
all zero plus a counter that starts at zero.

Some k values for my "a-=c; a^=rot(c,k); c+=b;" arrangement that
satisfy this are
4  6  8 16 19  4
9 15  3 18 27 15
14  9  3  7 17  3
Well, "9 15 3 18 27 15" didn't quite get 32 bits diffing
for "differ" defined as + with a one-bit base and a two-bit delta. I
used http://burtleburtle.net/bob/hash/avalanche.html to choose
the operations, constants, and arrangements of the variables.

This does not achieve avalanche. There are input bits of (a,b,c)
that fail to affect some output bits of (a,b,c), especially of a. The
most thoroughly mixed value is c, but it doesn't really even achieve
avalanche in c.

This allows some parallelism. Read-after-writes are good at doubling
the number of bits affected, so the goal of mixing pulls in the opposite
direction as the goal of parallelism. I did what I could. Rotates
seem to cost as much as shifts on every machine I could lay my hands
on, and rotates are much kinder to the top and bottom bits, so I used
rotates.

```

```

-----
*/
#define mix(a,b,c) \
{ \
    a -= c;  a ^= rot(c, 4);  c += b; \
    b -= a;  b ^= rot(a, 6);  a += c; \
    c -= b;  c ^= rot(b, 8);  b += a; \
    a -= c;  a ^= rot(c,16);  c += b; \
    b -= a;  b ^= rot(a,19);  a += c; \
    c -= b;  c ^= rot(b, 4);  b += a; \
}

/*
-----
final -- final mixing of 3 32-bit values (a,b,c) into c

Pairs of (a,b,c) values differing in only a few bits will usually
produce values of c that look totally different.  This was tested for
* pairs that differed by one bit, by two bits, in any combination
of top bits of (a,b,c), or in any combination of bottom bits of
(a,b,c).
* "differ" is defined as +, -, ^, or ~^.  For + and -, I transformed
the output delta to a Gray code (a^(a>>1)) so a string of 1's (as
is commonly produced by subtraction) look like a single 1-bit
difference.
* the base values were pseudorandom, all zero but one bit set, or
all zero plus a counter that starts at zero.

These constants passed:
14 11 25 16 4 14 24
12 14 25 16 4 14 24
and these came close:
4  8 15 26 3 22 24
10  8 15 26 3 22 24
11  8 15 26 3 22 24
-----
*/
#define hashfinal(a,b,c) \
{ \
    c ^= b; c -= rot(b,14); \
    a ^= c; a -= rot(c,11); \
    b ^= a; b -= rot(a,25); \
    c ^= b; c -= rot(b,16); \
}

```

```

    a ^= c; a -= rot(c,4); \
    b ^= a; b -= rot(a,14); \
    c ^= b; c -= rot(b,24); \
}

/*
-----
This works on all machines.  To be useful, it requires
-- that the key be an array of uint32_t's, and
-- that the length be the number of uint32_t's in the key

The function hashword() is identical to hashlittle() on little-endian
machines, and identical to hashbig() on big-endian machines,
except that the length has to be measured in uint32_ts rather than in
bytes.  hashlittle() is more complicated than hashword() only because
hashlittle() has to dance around fitting the key bytes into registers.
-----
*/
uint32_t hashword(
    const uint32_t *k,          /* the key, an array of uint32_t values */
    uint32_t      length,      /* the length of the key, in uint32_ts */
    uint32_t      initval)     /* the previous hash, or an arbitrary value */
{
    uint32_t a, b, c;

    /* Set up the internal state */
    a = b = c = 0xdeadbeef + (((uint32_t)length) << 2) + initval;

    /*----- handle most of the key */
    while (length > 3)
    {
        a += k[0];
        b += k[1];
        c += k[2];
        mix(a, b, c);
        length -= 3;
        k += 3;
    }

    /*----- handle the last 3 uint32_t's */
    switch (length)
    {
        /* all the case statements fall through */
        case 3: c += k[2];

```

```

        case 2: b += k[1];
        case 1: a += k[0];
            hashfinal(a, b, c);
        case 0:      /* case 0: nothing left to add */
            break;
    }
    /*----- report the result */
    return c;
}

```

```

uint64_t hashword64(
    const uint32_t *k,          /* the key, an array of uint32_t values */
    uint32_t length,           /* the length of the key, in uint32_ts */
    uint64_t initval)          /* the previous hash, or an arbitrary value */
{
    uint32_t high64;
    uint32_t low64;
    uint64_t retVal;
    uint32_t seed2;
    uint32_t i;

    high64 = (uint32_t)(initval >> 32);
    low64 = (uint32_t)(initval & 0x00000000ffffffff);
    high64 = hashword(k, length, high64);
    seed2 = high64; for (i = 0; i < length; i++) seed2 += k[i];
    low64 = hashword(k, length, seed2);

    retVal = (uint64_t)high64;
    retVal = (retVal << 32) + low64;
    return retVal;
}

```

/***** LEVENSHTTEIN EDIT DISTANCE CALCULATION *****/

```

////////////////////////////////////
///  Levenshtein edit distances  ///
////////////////////////////////////
#define MIN3(a, b, c) ((a) < (b) ? ((a) < (c) ? (a) : (c)) : ((b) < (c) ? (b) : (c)))
/*
int levenshtein(char *s1, char *s2) {
    unsigned int s1len, s2len, x, y, lastdiag, olddiag, retval;

```

```

uint32_t *column;

if ((s1 == NULL) || (s2 == NULL)) return MAXINT;
s1len = strlen(s1);
s2len = strlen(s2);
if ((s1len == 0) || (s2len == 0)) return MAXINT;

column = (uint32_t *)malloc(sizeof(uint32_t) * (s1len + 1));
if (column == NULL) return MAXINT;

//   if ((s1len > 39) || (s2len > 39)) printf("s1len = %i    s2len = %i    \n", (in

for (y = 1; y <= s1len; y++)
column[y] = y;
for (x = 1; x <= s2len; x++) {
column[0] = x;
for (y = 1, lastdiag = x - 1; y <= s1len; y++) {
olddiag = column[y];
column[y] = MIN3(column[y] + 1, column[y - 1] + 1, lastdiag + (s1[y - 1] == s2[x -
lastdiag = olddiag;
}
}
retval = column[s1len];
free(column);
return(retval);
}
*/
int levenshteinI(int *s1, uint32_t s1len, int *s2, uint32_t s2len) {
    unsigned int x, y, lastdiag, olddiag, retval;
    uint32_t *column;

    if ((s1 == NULL) || (s2 == NULL)) return MAXINT;
    if ((s1len <= 0) || (s2len <= 0)) return MAXINT;

    column = (uint32_t *)malloc(sizeof(uint32_t) * (s1len + 1));
    if (column == NULL) return MAXINT;

    //   if ((s1len <= 0) || (s2len <= 0) || (s1len > 39) || (s2len > 39)) printf("

    for (y = 1; y <= s1len; y++)
        column[y] = y;
    for (x = 1; x <= s2len; x++) {
        column[0] = x;

```

```

        for (y = 1, lastdiag = x - 1; y <= s1len; y++) {
            olddiag = column[y];
            column[y] = MIN3(column[y] + 1, column[y - 1] + 1, lastdiag + (s1[y - 1]
            lastdiag = olddiag;
        }
    }
    retval = column[s1len];
    free(column);
    return(retval);
}

```

/***** RETURN WORDS/NUMBERS FROM STRINGS *****/

```

char * strip(char * wvalue, char wsep, uint8_t widx) {
    uint8_t i, wflag;
    char *wpos, *wretstr, *wtemp;
    uint32_t wlen;

    wflag = 0;
    wpos = wvalue;
    for (i = 0; i <= widx; i++) {
        wpos = strchr(wvalue, wsep);
        if (wpos == NULL) {
            if (wflag == 0)
                wflag = 1;
            else
                return NULL;
        }
        else if (i < widx)
            wvalue = wpos + 1;
    }

    if (wflag) {
        wlen = (uint32_t)strlen(wvalue);
        wretstr = (char *)malloc(sizeof(char) * (wlen + 1));
        for (i = 0; i < wlen; i++) wretstr[i] = wvalue[i];
        wretstr[wlen] = 0;
    }
    else {
        wlen = (uint32_t)(wpos - wvalue);
        wretstr = (char *)malloc(sizeof(char) * (wlen + 1));
        for (i = 0; i < wlen; i++) wretstr[i] = wvalue[i];
    }
}

```



```

        wretstr[wlen] = 0;
    }
    wtemp = wretstr;
    wretstr = strtoupper(wretstr);
    free(wtemp);

    return wretstr;
}

uint32_t stripN(char * wvalue, char wsep, uint8_t widx) {
    char *wtemp;
    uint32_t wretval;

    wtemp = strip(wvalue, wsep, widx);

    wretval = ((wtemp != NULL) ? atoi(wtemp) : 0);
    if (wtemp != NULL) free(wtemp);

    return wretval;
}

////////////////////////////////////
// Pull out the value associated with the passed label //
////////////////////////////////////
char * stripLabel(char * wvalue, char * wlabel, char wsep) {

    int npos, npos2, wlen;
    uint8_t i;
    char *wretstr, *wtemp;

    npos = stringSearch(wvalue, wlabel);
    if (npos < 0) return NULL;
    wtemp = (char *)malloc(sizeof(char) * 2);
    wtemp[0] = wsep;
    wtemp[1] = 0;

    npos2 = stringSearchN(wvalue, wtemp, npos);
    if (npos2 == -1) npos2 = (int)strlen(wvalue);
    wlen = (int) (npos2 - npos - strlen(wlabel));
    wretstr = (char *)malloc(sizeof(char) * (wlen + 1));
    for (i = 0; i < wlen; i++) wretstr[i] = wvalue[npos + strlen(wlabel) + i];
    wretstr[wlen] = 0;
}

```

```

    wtemp = wretstr;
    wretstr = strtoupper(wretstr);
    free(wtemp);

    return wretstr;
}

uint32_t stripLabelN(char * wvalue, char * wlabel, char wsep) {
    char * wtemp;
    uint32_t wretval;

    wtemp = stripLabel(wvalue, wlabel, wsep);
    if (wtemp != NULL) {
        wretval = atoi(wtemp);
        free(wtemp);
    }
    else
        wretval = 0;
    return wretval;
}

uint32_t stripLabelT(char * wvalue, char * wlabel, char wsep) {
    char * wtemp;
    uint32_t wretval;

    wtemp = stripLabel(wvalue, wlabel, wsep);
    if (wtemp != NULL) {
        wretval = stringToTime(wtemp);
        free(wtemp);
    }
    else
        wretval = 0;
    return wretval;
}

/***** READ INPUT FROM A FILE *****/

/*****/
/* Read the next input value */
/*****/

```

```

char filastChar() { return lastChar; }

char * firead(FILE *dumpfile, char wsep) {
    char *woutput;
    unsigned int wchar, cnt, wloc;

    if (dumpfile == NULL) return NULL;

    wloc = ftell(dumpfile);

    // ignore blank lines
    wchar = fgetc(dumpfile);
    while ((wchar != EOF) && ((wchar == '\n') || (wchar == '\t') ||
        (wchar == wsep))) {
        wloc = ftell(dumpfile);
        wchar = fgetc(dumpfile);
    }

    cnt = 0;
    while ((wchar != '\n') && (wchar != '\t') && (wchar != EOF) &&
        (wchar != wsep)) {
        cnt++;
        wchar = fgetc(dumpfile);
    }
    lastChar = wchar;

    if (wchar == EOF) return NULL;

    fseek(dumpfile, wloc, SEEK_SET);
    woutput = (char *)malloc(sizeof(char) * (cnt + 1));
    cnt = 0;
    wchar = fgetc(dumpfile);
    while ((wchar != '\n') && (wchar != '\t') && (wchar != EOF) && (wchar != wsep))
        if (wchar < 0x20 || wchar > 0x7f || wchar == 0x60 || wchar == 0x27 ||
            wchar == 0x22) // 0x25 = % 0x27 = ' 0x60 = ' 0x22 = "
            woutput[cnt] = 0x20;
        else
            woutput[cnt] = wchar;
        cnt++;
        wchar = fgetc(dumpfile);
    }
    woutput[cnt] = 0;

```

```

    lastChar = wchar;
    return woutput;
}

char * firead(FILE *dumpfile, char wsep, uint8_t skipDoubleSep) {
    char *woutput;
    unsigned int wchar, cnt, wloc;

    if (dumpfile == NULL) return NULL;

    wloc = ftell(dumpfile);

    // ignore blank lines
    wchar = fgetc(dumpfile);
    while ((wchar != EOF) && ((wchar == '\n') || (skipDoubleSep &&
        ((wchar == '\t') || (wchar == wsep))))) {
        wloc = ftell(dumpfile);
        wchar = fgetc(dumpfile);
    }

    cnt = 0;
    while ((wchar != '\n') && (wchar != '\t') && (wchar != EOF) && (wchar != wsep))
        cnt++;
    wchar = fgetc(dumpfile);
}
lastChar = wchar;

if (wchar == EOF) return NULL;

fseek(dumpfile, wloc, SEEK_SET);
woutput = (char *)malloc(sizeof(char) * (cnt + 1));
cnt = 0;
wchar = fgetc(dumpfile);
while ((wchar != '\n') && (wchar != '\t') && (wchar != EOF) && (wchar != wsep))
    if (wchar < 0x20 || wchar > 0x7f || wchar == 0x60 || wchar == 0x27 ||
        wchar == 0x22) // 0x25 = % 0x27 = ' 0x60 = ' 0x22 = "
        woutput[cnt] = 0x20;
    else
        woutput[cnt] = wchar;
    cnt++;
    wchar = fgetc(dumpfile);
}
woutput[cnt] = 0;

```

```

    lastChar = wchar;
    return woutput;
}

char * firead_nl(FILE *dumpfile, char wsep) {
    char *woutput;
    unsigned int wchar, cnt, wloc;

    if (dumpfile == NULL) return NULL;

    wloc = ftell(dumpfile);

    wchar = fgetc(dumpfile);

    // ignore blank lines
    while ((wchar != EOF) && ((wchar == '\n') || (wchar == '\t') || (wchar == wsep)) {
        wloc = ftell(dumpfile);
        wchar = fgetc(dumpfile);
    }

    cnt = 0;
    while ((wchar != EOF) && (wchar != '\n')) {
        cnt++;
        wchar = fgetc(dumpfile);
    }

    lastChar = wchar;
    if (wchar == EOF) return NULL;

    fseek(dumpfile, wloc, SEEK_SET);
    woutput = (char *)malloc(sizeof(char) * (cnt + 1));
    cnt = 0;
    wchar = fgetc(dumpfile);
    while ((wchar != EOF) && (wchar != '\n')) {
        if (wchar < 0x20 || wchar > 0x7f || wchar == 0x60 || wchar == 0x27 ||
            wchar == 0x22) // 0x25 = % 0x27 = ' 0x60 = ' 0x22 = "
            woutput[cnt] = 0x20;
        else
            woutput[cnt] = wchar;
        cnt++;
        wchar = fgetc(dumpfile);
    }
}

```

```

woutput[cnt] = 0;

lastChar = wchar;
return woutput;
}

char * sread(char *wstring, char wsep, unsigned int wpos) {
    char *woutput;
    unsigned int wloc, wchar, cnt, strt, wsize, ctr;

    wloc = 0;
    cnt = 0;
    wchar = wstring[wloc];
    while (wloc <= strlen(wstring) && cnt < wpos) {
        // skip whitespace
        while ((wloc < strlen(wstring)) && ((wchar == ' ') || (wchar == '\t') ||
            (wchar == wsep))) {
            wchar = wstring[++wloc];
        }

        // skip fields less than desired
        while ((wchar != ' ') && (wchar != '\t') &&
            (wloc <= strlen(wstring)) && (wchar != wsep) && cnt < wpos) {
            wchar = wstring[++wloc];
        }

        cnt++;

        if (wloc > strlen(wstring)) return NULL;
    }

    // skip whitespace
    while ((wloc <= strlen(wstring)) && ((wchar == ' ') || (wchar == '\t') ||
        (wchar == wsep))) {
        wchar = wstring[++wloc];
    }

    // calculate the size and then create a string
    strt = wloc;
    wsize = 0;
    while ((wchar != ' ') && (wchar != '\t') && (wloc <= strlen(wstring)) &&
        (wchar != wsep)) {
        wchar = wstring[++wloc];
    }

```

```

        wsize++;
    }
    wloc = strt;
    woutput = (char *)malloc(sizeof(char) * (wsize + 2));

    // load the field into the string and return it
    ctr = 0;
    wchar = wstring[wloc];
    while ((wchar != ' ') && (wchar != '\t') &&
           (wloc <= strlen(wstring)) && (wchar != wsep)) {
        wchar = wstring[wloc++];
        if (wchar < 0x20 || wchar > 0x7f || wchar == 0x60 ||
            wchar == 0x27 || wchar == 0x22) // 0x25 = % 0x27 = ' 0x60 = ' 0x22 = "
            woutput[ctr++] = 0x20;
        else
            woutput[ctr++] = wchar;
    }
    woutput[--ctr] = 0;

    return woutput;
}

```

```

////////////////////////////////////
// printable                                //
//                                           //
// Determines if the passed character is printable //
//                                           //
// Receives:                                //
//     a character field                    //
// Returns:                                //
//     1 if the character is between 32 and 127 ascii //
//     0 otherwise                          //
////////////////////////////////////
uint8_t printable(char c) { return (c >= 32 && c <= 127); }

```

```

////////////////////////////////////
// get_cfield                                //
//                                           //
// Wrapper to return the next character field from a .csv file //
//                                           //
// Receives:                                //
//     file handle of the .csv file        //

```

```

// Returns: //
//      a pointer to a malloc'd character string //
// If there's nothing to return, it returns a 1 byte string where [0] = 0 //
// //
char * get_cfield(FILE *ifile) { return get_field(ifile); }

// //
// get_ifield //
// //
// Wrapper to return the next integer field from a .csv file //
// //
// Receives: //
//      file handle of the .csv file //
// Returns: //
//      an integer value //
//      0 if the next value isn't an interger or if there's nothing left //
//      to return from the .csv file //
// //

uint32_t get_ifield(FILE *ifile) {
    char *wstring;
    uint32_t retval;

    wstring = get_field(ifile);
    retval = atoi(wstring);
    free(wstring);
    return retval;
} // get_ifield

double get_ffield(FILE *ifile) {
    char *wstring;
    double retval;

    wstring = get_field(ifile);
    retval = atof(wstring);
    free(wstring);
    return retval;
} // get_ffield

void getEoln(FILE *ifile) {

    uint8_t c = getc(ifile);
    while (!feof(ifile) && (c != '\n')) c = getc(ifile);

```



```

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// get_field                                                                    //
//                                                                    //
// Internal function to retrieve the next character field from a .csv file //
//                                                                    //
// Receives:                                                                    //
//     file handle of the .csv file                                            //
// Returns:                                                                    //
//     a pointer to a malloc'd character string                              //
// Character strings are expected to be quoted if they contain commas        //
// Only printable characters (ascii 32 - 127) are included in the returned    //
//     string.                                                                    //
// If there's nothing to return, it returns a 1 byte string where [0] = 0    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char * get_field(FILE *ifile) {
    uint32_t i;
    uint8_t is_quote;    // 1 if an odd number of quotes has been seen, 0 otherwise
    uint8_t done;        // 1 if the a comma or newline has been seen
    uint8_t c;           // character retrieved from the file
    uint32_t ctr;        // counter of number of character retrieved
    char wtext[5000];    // buffer to hold string while processing
    char *retval;        // pointer to a malloc'd string to return to the caller

        // Initialize variables
    is_quote = 0;
    ctr = 0;

    // Retrieve the first character.
    // If it's a comma or newline, we're done
    c = getc(ifile);
    done = (c == ',') || (c == '\n');

    // While not done (i.e., no comma or newline) and there's more to read
    while (!feof(ifile) && (!done)) {

        // If a quote is encountered, flip the is_quote flag (but don't print it)
        if (c == '"')
            is_quote = !is_quote;

        // Otherwise, if the character is printable, add it to the string
        else {

```

```

        if (printable(c))
            wtext[ctr++] = c;
    }

    // Get the next character
    c = getc(ifile);

    // If the character is a comma or newline, we're done
    done = ((c == ',') || (c == '\n')) && (!is_quote);

    // If more than 5000 characters have been read, we're done
    if (ctr == 5000) break;
}

// Initialize the malloc'd variable to return
retval = (char *)malloc(sizeof(char) * (ctr + 1));
if (retval == NULL) return NULL;

// Load the malloc'd string with the string that was read
for (i = 0; i < ctr; i++) retval[i] = wtext[i]; retval[i] = 0;

// Return a pointer to the malloc'd string
return retval;
} //get_field

```

```

//////////////////////////////////////////////////////////////////
// days                                                    //
//                                                         //
// Calculates the number of days elapsed in the year      //
//                                                         //
// Receives:                                              //
//     a valid month, day, year combination in time tm   //
//     format (i.e., 0 - 11, 0 - 31, years past 1900)    //
// Returns:                                               //
//     the number of days elapsed in the specified year   //
//////////////////////////////////////////////////////////////////
uint32_t days(uint32_t wmon, uint32_t wyear, uint32_t wdays) {
    uint32_t i;

    wyear += 1900;

    for (i = 0; i < wmon; i++) {

```

```

        switch (i) {
        case 0: wdays += 31; break;
        case 1: if (wyear % 4 != 0) { wdays += 28; break; }
                if (wyear % 400 == 0) { wdays += 29; break; }
                if (wyear % 100 == 0) { wdays += 28; break; }
                wdays += 29; break;
        case 2: wdays += 31; break;
        case 3: wdays += 30; break;
        case 4: wdays += 31; break;
        case 5: wdays += 30; break;
        case 6: wdays += 31; break;
        case 7: wdays += 31; break;
        case 8: wdays += 30; break;
        case 9: wdays += 31; break;
        case 10: wdays += 30; break;
        case 11: wdays += 31; break;
        }
    }
    return wdays;
} //days

/////////////////////////////////////////////////////////////////
// convertToUnixTime                                     //
//                                                         //
// Converts an integer date in the form of MMDDYYYY to    //
// the number of seconds since 1/1/1970 (Unix Time)       //
//                                                         //
// Receives:                                              //
//     an integer representing the date in MMDDYYYY form//
// Returns:                                              //
//     a 64 bit integer representing the number of seconds //
//     since 1/1/1970 (Unix time)                        //
/////////////////////////////////////////////////////////////////
uint64_t convertToUnixTime(uint32_t wdate) {
    struct tm *timeinfo;
    uint32_t wtemp;
    uint64_t rdate;

    timeinfo = (struct tm *)malloc(sizeof(struct tm) * 1);
    if (timeinfo == NULL) return 0;

    timeinfo->tm_hour = 0;
    timeinfo->tm_min = 0;

```

```

    timeinfo->tm_sec = 0;
    timeinfo->tm_mon = ((uint32_t)(wdate / 1000000.0));
    wtemp = wdate - (timeinfo->tm_mon * 1000000);
    timeinfo->tm_mday = ((uint32_t)(wtemp / 10000.0));
    wtemp = wtemp - (((uint32_t)(wtemp / 10000.0)) * 10000);
    timeinfo->tm_year = wtemp;
    timeinfo->tm_isdst = isDaylightSavingsTime;
    timeinfo->tm_wday = 0;
    timeinfo->tm_yday = days(timeinfo->tm_mon, timeinfo->tm_year, timeinfo->tm_mday);
    timeinfo->tm_mon -= 1;
    timeinfo->tm_year -= 1900;

    rdate = (uint64_t)mkttime(timeinfo);
    free(timeinfo);
    return rdate;
}

uint32_t convertToUnixSeconds(uint32_t wtime) {
    uint32_t whours, wminutes, wseconds, wtemp;

    whours = ((uint32_t)(wtime / 10000.0));
    wtemp = wtime - (whours * 10000);
    wminutes = ((uint32_t)(wtemp / 100.0));
    wseconds = wtemp - (wminutes * 100);

    return (whours * 3600 + wminutes * 60 + wseconds);
}

/*****CONVERT DATE/TIME FORMATS*****/

struct tm * nanoToTime(uint64_t itime) {
    struct tm *otime;
    uint64_t nanodays, wdays, nanohours, nanomins, nanosecs, wadj;

    //    printf("itime = %llx\n", itime);

    otime = (tm *)malloc(sizeof(struct tm) * 1);
    otime->tm_isdst = -1;
    otime->tm_year = 1601;
    otime->tm_mon = 0;

    otime->tm_wday = 0;

```

```

nanodays = 864000000000; // Number of 100 nanoseconds in a day
nanohours = 360000000000; // Number of 100 nanoseconds in a hour
nanomins = 6000000000; // Number of 100 nanoseconds in a minute
nanosecs = 100000000; // Number of 100 nanoseconds in a second
wdays = nanodays * 365;
while (itime > wdays) {
    itime -= wdays;
    otime->tm_year++;
    wadj = (((otime->tm_year % 4 == 0) && ((otime->tm_year % 100 != 0) ||
        (otime->tm_year % 400 == 0))) ? 366 : 365);
    wdays = nanodays * wadj;
}
otime->tm_year -= 1900;

wdays = nanodays * 31;
while (itime > wdays) {
    itime -= wdays;
    otime->tm_mon++;
    switch (otime->tm_mon) {
        case 1: wdays = nanodays * (((otime->tm_year % 4 == 0) &&
            ((otime->tm_year % 100 != 0) || (otime->tm_year % 400 == 0))) ? 29
            otime->tm_yday = 31;
            break;
        case 2: wdays = nanodays * 31;
            otime->tm_yday += (((otime->tm_year % 4 == 0) &&
                ((otime->tm_year % 100 != 0) || (otime->tm_year % 400 == 0))) ? 29
            break;
        case 3: wdays = nanodays * 30; otime->tm_yday += 31; break;
        case 4: wdays = nanodays * 31; otime->tm_yday += 30; break;
        case 5: wdays = nanodays * 30; otime->tm_yday += 31; break;
        case 6: wdays = nanodays * 31; otime->tm_yday += 30; break;
        case 7: wdays = nanodays * 31; otime->tm_yday += 31; break;
        case 8: wdays = nanodays * 30; otime->tm_yday += 31; break;
        case 9: wdays = nanodays * 31; otime->tm_yday += 30; break;
        case 10: wdays = nanodays * 30; otime->tm_yday += 31; break;
        case 11: wdays = nanodays * 31; otime->tm_yday += 30; break;
    }
}

otime->tm_mday = (int)(itime / nanodays);
otime->tm_yday += otime->tm_mday;
itime -= otime->tm_mday * nanodays;
otime->tm_hour = (int)(itime / nanohours);

```

```

        itime -= otime->tm_hour * nanohours;
        otime->tm_min = (int)(itime / nanomins);
        itime -= otime->tm_min * nanomins;
        otime->tm_sec = (int)(itime / nanosecs);
        otime->tm_mday++;
        return otime;
}

uint32_t stringToTime(char * stime) {
    char tstTime[4];
    uint32_t retval;

    retval = 0;

    tstTime[0] = stime[0];
    tstTime[1] = stime[1];
    tstTime[2] = stime[2];
    tstTime[3] = stime[3];

    if (atoi(tstTime) >= 1900 && atoi(tstTime) < 2100)
        retval = stringToTime1(stime);

    tstTime[0] = stime[6];
    tstTime[1] = stime[7];
    tstTime[2] = stime[8];
    tstTime[3] = stime[9];

    if (retval == 0 && atoi(tstTime) >= 1900 && atoi(tstTime) < 2100)
        retval = stringToTime2(stime);

    if (retval <= 0) retval = stringToTime3(stime);

    return retval;
}

uint32_t stringToTime1(char * stime) { //12-31-2013 11:59:00

    struct tm *otime;
    uint8_t xmonth;
    char syear[5], smonth[3], sdom[3], shour[3], smin[3], ssec[3], sdow[2];

    syear[0] = stime[6]; syear[1] = stime[7]; syear[2] = stime[8];

```

```

syear[3] = stime[9]; syear[4] = 0;
smonth[0] = stime[0]; smonth[1] = stime[1]; smonth[2] = 0;
sdom[0] = stime[3]; sdom[1] = stime[4]; sdom[2] = 0;
shour[0] = stime[11]; shour[1] = stime[12]; shour[2] = 0;
smin[0] = stime[14]; smin[1] = stime[15]; smin[2] = 0;
ssec[0] = stime[17]; ssec[1] = stime[18]; ssec[2] = 0;
sdow[0] = stime[25]; sdow[1] = 0;

otime = (tm *)malloc(sizeof(struct tm) * 1);
otime->tm_isdst = -1;
otime->tm_year = atoi(syear) - 1900;
otime->tm_mon = atoi(smonth) - 1;
otime->tm_wday = atoi(sdow);
otime->tm_mday = atoi(sdom);
otime->tm_hour = atoi(shour);
otime->tm_min = atoi(smin);
otime->tm_sec = atoi(ssec);

xmonth = otime->tm_mon;
otime->tm_yday = 0;
while (xmonth > 0) {
    switch (xmonth) {
        case 1: otime->tm_yday = 31;
            break;
        case 2: otime->tm_yday += (((otime->tm_year % 4 == 0) &&
            ((otime->tm_year % 100 != 0) ||
            (otime->tm_year % 400 == 0))) ? 29 : 28);
            break;
        case 3: otime->tm_yday += 31; break;
        case 4: otime->tm_yday += 30; break;
        case 5: otime->tm_yday += 31; break;
        case 6: otime->tm_yday += 30; break;
        case 7: otime->tm_yday += 31; break;
        case 8: otime->tm_yday += 31; break;
        case 9: otime->tm_yday += 30; break;
        case 10: otime->tm_yday += 31; break;
        case 11: otime->tm_yday += 30; break;
    }
    xmonth--;
}
otime->tm_yday += otime->tm_mday;

return ((uint32_t)(mktime(otime)));

```

```

}

uint32_t stringToTime2(char * stime) { // 2013-03-22 15:19:58
    struct tm *otime;
    uint8_t xmonth;
    char syear[5], smonth[3], sdom[3], shour[3], smin[3], ssec[3], sdow[2];

    syear[0] = stime[0]; syear[1] = stime[1]; syear[2] = stime[2];
    syear[3] = stime[3]; syear[4] = 0;
    smonth[0] = stime[5]; smonth[1] = stime[6]; smonth[2] = 0;
    sdom[0] = stime[8]; sdom[1] = stime[9]; sdom[2] = 0;
    shour[0] = stime[11]; shour[1] = stime[12]; shour[2] = 0;
    smin[0] = stime[14]; smin[1] = stime[15]; smin[2] = 0;
    ssec[0] = stime[17]; ssec[1] = stime[18]; ssec[2] = 0;
    sdow[0] = 0; sdow[1] = 0;

    otime = (tm *)malloc(sizeof(struct tm) * 1);
    otime->tm_isdst = -1;
    otime->tm_year = atoi(syear) - 1900;
    otime->tm_mon = atoi(smonth) - 1;
    otime->tm_wday = atoi(sdow);
    otime->tm_mday = atoi(sdom);
    otime->tm_hour = atoi(shour);
    otime->tm_min = atoi(smin);
    otime->tm_sec = atoi(ssec);

    xmonth = otime->tm_mon;
    otime->tm_yday = 0;
    while (xmonth > 0) {
        switch (xmonth) {
            case 1: otime->tm_yday = 31;
                    break;
            case 2: otime->tm_yday += (((otime->tm_year % 4 == 0) &&
                                         ((otime->tm_year % 100 != 0) ||
                                          (otime->tm_year % 400 == 0))) ? 29 : 28);
                    break;
            case 3: otime->tm_yday += 31; break;
            case 4: otime->tm_yday += 30; break;
            case 5: otime->tm_yday += 31; break;
            case 6: otime->tm_yday += 30; break;
            case 7: otime->tm_yday += 31; break;
            case 8: otime->tm_yday += 31; break;
            case 9: otime->tm_yday += 30; break;
        }
    }
}

```



```

        case 10: otime->tm_yday += 31; break;
        case 11: otime->tm_yday += 30; break;
    }
    xmonth--;
}
otime->tm_yday += otime->tm_mday;

return ((uint32_t)(mktime(otime)));
}

uint32_t stringToTime3(char * stime) { // Wed Aug 13 13:23:35 2016 UTC -- variable
    struct tm *otime;
    uint8_t xmonth, imonth, idow;
    char *wmonth, *wday, *whour, *wminute, *wsecond, *wyear, *wdow;

    imonth = xmonth = idow = 0;
    wdow = sread(stime, ':', 0);
    wmonth = sread(stime, ':', 1);
    wday = sread(stime, ':', 2);
    whour = sread(stime, ':', 3);
    wminute = sread(stime, ':', 4);
    wsecond = sread(stime, ':', 5);
    wyear = sread(stime, ':', 6);
    if (strstr("Jan", wmonth) != NULL) imonth = 0;
    else if (strstr("Feb", wmonth) != NULL) imonth = 1;
    else if (strstr("Mar", wmonth) != NULL) imonth = 2;
    else if (strstr("Apr", wmonth) != NULL) imonth = 3;
    else if (strstr("May", wmonth) != NULL) imonth = 4;
    else if (strstr("Jun", wmonth) != NULL) imonth = 5;
    else if (strstr("Jul", wmonth) != NULL) imonth = 6;
    else if (strstr("Aug", wmonth) != NULL) imonth = 7;
    else if (strstr("Sep", wmonth) != NULL) imonth = 8;
    else if (strstr("Oct", wmonth) != NULL) imonth = 9;
    else if (strstr("Nov", wmonth) != NULL) imonth = 10;
    else if (strstr("Dec", wmonth) != NULL) imonth = 11;
    else imonth = 0;
    if (strstr("Sun", wdow) != NULL) idow = 0;
    else if (strstr("Mon", wdow) != NULL) idow = 1;
    else if (strstr("Tue", wdow) != NULL) idow = 2;
    else if (strstr("Wed", wdow) != NULL) idow = 3;
    else if (strstr("Thu", wdow) != NULL) idow = 4;
    else if (strstr("Fri", wdow) != NULL) idow = 5;
    else if (strstr("Sat", wdow) != NULL) idow = 6;

```

```

else idow = 0;

otime = (tm *)malloc(sizeof(struct tm) * 1);
otime->tm_isdst = -1;
otime->tm_year = atoi(wyear) - 1900;
otime->tm_mon = imonth;
otime->tm_wday = idow;
otime->tm_mday = atoi(wday);
otime->tm_hour = atoi(whour);
otime->tm_min = atoi(wminute);
otime->tm_sec = atoi(wsecond);

free(wdow); free(wmonth); free(wday); free(whour); free(wminute); free(wsecond);
xmonth = otime->tm_mon;
otime->tm_yday = 0;
while (xmonth > 0) {
    switch (xmonth) {
        case 1: otime->tm_yday = 31;
            break;
        case 2: otime->tm_yday += (((otime->tm_year % 4 == 0) &&
            ((otime->tm_year % 100 != 0) ||
            (otime->tm_year % 400 == 0))) ? 29 : 28);
            break;
        case 3: otime->tm_yday += 31; break;
        case 4: otime->tm_yday += 30; break;
        case 5: otime->tm_yday += 31; break;
        case 6: otime->tm_yday += 30; break;
        case 7: otime->tm_yday += 31; break;
        case 8: otime->tm_yday += 31; break;
        case 9: otime->tm_yday += 30; break;
        case 10: otime->tm_yday += 31; break;
        case 11: otime->tm_yday += 30; break;
    }
    xmonth--;
}
otime->tm_yday += otime->tm_mday;

return ((uint32_t)(mktime(otime)));
}

// Add a \ to the end of the path if not present
void fixpath(char *path) {
    if (path[strlen(path) - 1] == L'\\') path[strlen(path) - 1] = 0;
}

```

}

};

8.6 queues

Header.

```
#pragma once
#include <stdlib.h>
//#include <stdio.h>

template <class T>
class queueT {

    typedef struct queue_entry_type {
        T *ptr;
        struct queue_entry_type *prev;
        struct queue_entry_type *next;
    } qentry;

    qentry *front;
    qentry *rear;
    unsigned int count;

public:
    queueT() { front = NULL;   rear = NULL; count = 0; }
    ~queueT() {
        qentry *wptr, *wtail;
        wptr = front;
        while (wptr != NULL) {
            wtail = wptr;
            wptr = wptr->prev;
            free(wtail);
        }
    }
    qentry *initializeIterator() { return front; }
    qentry *iterate(void *wptr) {
        if (wptr == NULL) return NULL;
        return ((qentry *)wptr)->prev;
    }
    T *qvalue(void *wptr) {
        if (wptr == NULL) return NULL;
        return ((qentry *)wptr)->ptr;
    }
    void qpush(T *wptr) {
        qentry *wentry;
```

```

    wentry = (qentry *)malloc(sizeof(qentry) * 1);
    wentry->ptr = wptr;
    wentry->prev = NULL;
    wentry->next = rear;
    if (rear != NULL) rear->prev = wentry;
    rear = wentry;
    if (front == NULL) front = wentry;
    count++;
}

T *qpop() {
    qentry *retval;
    T *retPtr;
    retval = front;
    if (front == NULL) return NULL;
    if (front != NULL) front = front->prev;
    if (front != NULL) front->next = NULL;
    if (front == NULL) rear = NULL;

    retPtr = retval->ptr;
    free(retval);

    if (count > 0) count--;
    return retPtr;
}

T *qpopRear() {
    qentry *retval;
    T *retPtr;
    retval = rear;
    if (rear == NULL) return NULL;
    if (rear != NULL) rear = rear->next;
    if (rear != NULL) rear->prev = NULL;
    if (rear == NULL) front = NULL;

    retPtr = retval->ptr;
    free(retval);

    if (count > 0) count--;
    return retPtr;
}

void qclear() {

```

```

    qentry *wptr;
    while (front != NULL) {
        wptr = front;
        front = front->prev;
        free(wptr);
    }
    count = 0;
    rear = NULL;
}

unsigned char qempty() { if (front == NULL) return 1; return 0; }

unsigned int qcount() { return count; }

bool exists(T * wptr) {
    qentry *witer;
    witer = front;
    while (witer != NULL) {
        if (witer->ptr == wptr) return true;
        witer = witer->prev;
    }
    return false;
}

bool qerase(T *wptr) {
    qentry *witer, *wprev;

    if (wptr == NULL) return true;
    witer = rear;
    wprev = NULL;
    while (witer != NULL) {
        if (witer->ptr == wptr) {
            if (wprev != NULL)
                wprev->next = witer->next;
            else
                rear = witer->next;

            if (witer->next != NULL)
                witer->next->prev = wprev;
            else
                front = wprev;
            free(witer);
            return true;
        }
        wprev = witer;
        witer = witer->next;
    }
    return false;
}

```

```

        }
        wprev = witer;
        witer = witer->next;
    }

    // not found
    return false;
}

};

```

8.7 stacks

Header.

```

#pragma once
#include <stdlib.h>
//#include <stdio.h>

template <class T>
class stackT {

    typedef struct stack_entry_type {
        T *ptr;
        struct stack_entry_type *prev;
        struct stack_entry_type *next;
    } sentry;

    sentry *front;
    sentry *rear;

public:
    stackT() { front = NULL;   rear = NULL; }
    ~stackT() {
        sentry *wptr, *wtail;
        wptr = front;
        while (wptr != NULL) {
            wtail = wptr;
            wptr = wptr->prev;
            free(wtail);
        }
    }
    sentry *getTop() { return front; }

```

```

sentry *getBottom() { return rear; }
sentry *iterate(void *wptr) {
    if (wptr == NULL) return NULL;
    return ((sentry *)wptr)->next;
}
sentry *revIterate(void *wptr) {
    if (wptr == NULL) return NULL;
    return ((sentry *)wptr)->prev;
}
T *svalue(void *wptr) {
    if (wptr == NULL) return NULL;
    return ((sentry *)wptr)->ptr;
}
void spush(T *wptr) {
    sentry *wentry;

    wentry = (sentry *)malloc(sizeof(sentry) * 1);
    wentry->ptr = wptr;
    wentry->prev = front;
    wentry->next = NULL;
    if (front != NULL) front->next = wentry;
    front = wentry;
    if (rear == NULL) front = wentry;
}

T *spop() {
    T *wptr;
    sentry *retval;

    retval = front;
    if (front == NULL) return NULL;
    if (front != NULL) front = front->prev;
    if (front != NULL) front->next = NULL;
    if (front == NULL) rear = NULL;

    wptr = retval->ptr;
    free(retval);
    return wptr;
}

void sclear() {
    sentry *siter, *stail;
    siter = front;

```



```

        while (siter != NULL) {
            stail = siter;
            siter = siter->prev;
            free(stail);
        }
        front = NULL;
        rear = NULL;
    }

    bool exists(T * wptr) {
        sentry *witer;
        witer = front;
        while (witer != NULL) {
            if (witer->ptr == wptr) return true;
            witer = witer->prev;
        }
        return false;
    }

    unsigned char empty() { if (front == NULL) return 1; return 0; }
};

```

8.8 lists

Header.

```

#pragma once
#include <stdlib.h>
//#include <stdio.h>

template <class T>
class listT {

    typedef struct list_entry_type {
        T *ptr;
        struct list_entry_type *prev;
        struct list_entry_type *next;
    } lentry;

    lentry *front;
    lentry *rear;
    unsigned int wcount;

public:

```

```

listT() { front = NULL; rear = NULL; wcount = 0; }
~listT() {
    lentry *wptr, *wtail;
    wptr = front;
    while (wptr != NULL) {
        wtail = wptr;
        wptr = wptr->prev;
        free(wtail);
    }
}

lentry *getFront() { return front; }
lentry *getRear() { return rear; }
unsigned int getCount() { return wcount; }
lentry *iterate(void *wptr) {
    if (wptr == NULL) return NULL;
    if (((lentry *)wptr)->next == NULL) return NULL;
    return ((lentry *)wptr)->next;
}
lentry *revIterate(void *wptr) {
    if (wptr == NULL) return NULL;
    if (((lentry *)wptr)->prev == NULL) return NULL;
    return ((lentry *)wptr)->prev;
}
T *lvalue(void *wptr) {
    if (wptr == NULL) return NULL;
    return ((lentry *)wptr)->ptr;
}

unsigned char lempy() { if (front == NULL) return 1; return 0; }

void lclear() {
    lentry *liter, *ltail;
    liter = front;
    while (liter != NULL) {
        ltail = liter;
        liter = liter->prev;
        free(ltail);
    }
    front = NULL;
    rear = NULL;
    wcount = 0;
}

```

```

listT *copyPrimitive() {
    listT *wnewList;
    lentry *woldList, *wnewItem, *wtail;
    T *wval;
    wnewList = new listT<T>;
    woldList = front;
    wtail = NULL;
    while (woldList != NULL) {
        wnewItem = (lentry *)malloc(sizeof(lentry) * 1);
        wnewItem->prev = NULL;
        wnewItem->next = NULL;
//        wval = new T(woldList->ptr);
        wval = (T *)malloc(sizeof(T) * 1);
        *wval = *(woldList->ptr);
        wnewItem->ptr = wval;
        (wnewList->wcount)++;

        if (wnewList->front == NULL) wnewList->front = wnewItem;
        if (wtail != NULL) {
            wtail->prev = wnewItem;
            wnewItem->next = wtail;
        }
        wtail = wnewItem;
        woldList = woldList->prev;
    }
    wnewList->rear = wtail;
    return wnewList;
}

listT *copyClass() {
    listT *wnewList;
    lentry *woldList, *wnewItem, *wtail;
    T *wval;
    wnewList = new listT<T>;
    woldList = front;
    wtail = NULL;
    while (woldList != NULL) {
        wnewItem = (lentry *)malloc(sizeof(lentry) * 1);
        wnewItem->prev = NULL;
        wnewItem->next = NULL;
        wval = new T(woldList->ptr);
//        wval = (T *)malloc(sizeof(T) * 1);
//        *wval = *(woldList->ptr);

```

```

    wnewItem->ptr = wval;
    (wnewList->wcount)++;

    if (wnewList->front == NULL) wnewList->front = wnewItem;
    if (wtail != NULL) {
        wtail->prev = wnewItem;
        wnewItem->next = wtail;
    }
    wtail = wnewItem;
    woldList = woldList->prev;
}
wnewList->rear = wtail;
return wnewList;
}

void insertFront(T *wptr) {
    lentry *wentry;
    wentry = (lentry *)malloc(sizeof(lentry) * 1);
    wentry->ptr = wptr;
    wentry->next = NULL;
    wentry->prev = NULL;
    if (front == NULL) { front = wentry; rear = wentry; wcount = 1; return; }
    front->next = wentry; wentry->prev = front; front = wentry;
    wcount++;
    return;
}

void loadRear(listT<T> *wlist) {
    void *liter;
    if (wlist == NULL) return;
    if (wlist->lempty()) return;
    for (liter = wlist->getFront(); liter != NULL;
         liter = wlist->revIterate(liter)) {
        insertRear(wlist->lvalue(liter));
    }
}

void insertRear(T *wptr) {
    lentry *wentry;
    wentry = (lentry *)malloc(sizeof(lentry) * 1);
    wentry->ptr = wptr;
    wentry->next = NULL;

```

```

    wentry->prev = NULL;
    if (rear == NULL) { front = wentry; rear = wentry; wcount = 1; return; }
    rear->prev = wentry; wentry->next = rear; rear = wentry;
    wcount++;
    return;
}

bool exists(T * wptr) {
    lentry *witer;
    witer = front;
    while (witer != NULL) {
        if (witer->ptr == wptr) return true;
        witer = witer->prev;
    }
    return false;
}

bool lerase(void *liter) {
    lentry *witer;

    if (liter == NULL) return false;

    witer = (lentry *)liter;
    if (witer->prev != NULL) witer->prev->next = witer->next;
    if (witer->next != NULL) witer->next->prev = witer->prev;
    if (front == witer) front = witer->next;
    if (rear == witer) rear = witer->prev;
    wcount--;
    free(witer);

    return true;
}

bool lerase(T *wptr) {
    lentry *witer, *wprev;

    if (wptr == NULL) return true;
    witer = rear;
    wprev = NULL;
    while (witer != NULL) {
        if (witer->ptr == wptr) {
            if (wprev != NULL)
                wprev->next = witer->next;

```

```

        else
            rear = witer->next;

        if (witer->next != NULL)
            witer->next->prev = wprev;
        else
            front = wprev;
        wcount--;
        free(witer);
        return true;
    }
    wprev = witer;
    witer = witer->next;
}

// not found
return false;
}

void insertSort(T *wptr, bool(*flessThan)(const T *a, const T *b)) {
    lentry *wentry;
    lentry *witer;

    if (wptr == NULL) return;
    wentry = (lentry *)malloc(sizeof(lentry) * 1);
    wentry->ptr = wptr;
    wentry->next = NULL;
    wentry->prev = NULL;
    wcount++;

    // Case 1: list is empty
    if (rear == NULL) {
        front = wentry;
        rear = wentry;
        return;
    }

    // Case 2: it fits somewhere in the list (but not the front)
    witer = rear;
    while (witer != NULL) {
        if (flessThan(witer->ptr, wentry->ptr)) {
            wentry->next = witer;
            wentry->prev = witer->prev;

```

```

        if (witer->prev != NULL)
            witer->prev->next = wentry;
        else
            rear = wentry;
        witer->prev = wentry;
        return;
    }
    witer = witer->next;
}

// Case 3: it fits at the front
wentry->prev = front;
front->next = wentry;
front = wentry;

return;
}

void insertUnique(T *wptr, bool(*flessThan)(const T *a, const T *b)) {
    lentry *wentry;
    lentry *witer;

    if (wptr == NULL) return;
    wentry = (lentry *)malloc(sizeof(lentry) * 1);
    wentry->ptr = wptr;
    wentry->next = NULL;
    wentry->prev = NULL;
    wcount++;

    // Case 1: list is empty
    if (rear == NULL) {
        front = wentry;
        rear = wentry;
        return;
    }

    // Case 2: it fits somewhere in the list (but not the front)
    witer = rear;
    while (witer != NULL) {
        if (!flessThan(witer->ptr, wentry->ptr) &&
            !flessThan(witer->ptr, wentry->ptr)) { // equal
            free(wentry);

```

```

        return;
    }
    if (flessThan(witer->ptr, wentry->ptr)) {
        wentry->next = witer;
        wentry->prev = witer->prev;
        if (witer->prev != NULL)
            witer->prev->next = wentry;
        else
            rear = wentry;
        witer->prev = wentry;
        return;
    }
    witer = witer->next;
}

// Case 3: it fits at the front
wentry->prev = front;
front->next = wentry;
front = wentry;

return;
}

void insert(void *wptr, T *wval) {
    lentry *wentry;
    lentry *wprev;
    wprev = (lentry *)wptr;

    if (wptr == NULL) return;
    wentry = (lentry *)malloc(sizeof(lentry) * 1);
    wentry->ptr = wval;
    wentry->next = wprev->next;
    wprev->next->prev = wentry;
    wentry->prev = wprev;
    wprev->next = wentry;
    wcount++;

    return;
}

};

```


Bibliography

1. Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering*, pages 3–14. IEEE, 1995.
2. Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
3. H Karamath Ali and DI George Amalarethinam. Recognition of concurrent and interleaved activities in a smart environment using finite automata. *International Journal of Computer Applications*, 88(16):41–45, 2014.
4. Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 35:58–67, 2014.
5. Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.
6. Andrea Baraldi and Palma Blonda. A survey of fuzzy clustering algorithms for pattern recognition I. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(6):778–785, 1999.
7. Andrea Baraldi and Palma Blonda. A survey of fuzzy clustering algorithms for pattern recognition II. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(6):786–801, 1999.
8. Nicole Beebe. Digital forensic research: the good, the bad and the unaddressed. In *Advances in Digital Forensics V*, pages 17–36. Springer, 2009.
9. Gill Bejerano and Golan Yona. Modeling protein families using probabilistic suffix trees. In *Proceedings of the Third Annual International Conference on Computational Molecular Biology*, pages 15–24. ACM, 1999.
10. A.W. Biermann. On the synthesis of finite state machines from samples of their behavior. *IEEE Transactions on Computers*, C-21(6):592–597, 1972.
11. David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
12. Adam Boileau. Hit by a bus: physical access attacks with firewire, 2006. Ruxcon.
13. Florian P Buchholz and Courtney Falk. Design and implementation of zeitline: a forensic timeline editor. In *Proceedings of the 2005 Digital Forensics Research Workshop*, pages 1–7, 2005.

14. Cristian Bucilă, Johannes Gehrke, Daniel Kifer, and Walker White. Dualminer: A dual-pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery*, 7(3):241–272, 2003.
15. Brian Carrier. File activity timelines, 2003. Available online on Sep-2006 at http://www.sleuthkit.org/sleuthkit/docs/ref_timeline.html.
16. Brian Carrier. The sleuth kit and autopsy: forensics tools for linux and other unixes, 2015. Available online on May-2016 at <http://www.sleuthkit.org>.
17. Brian Carrier and Eugene H Spafford. An event-based digital forensic investigation framework. In *Proceedings of the 2004 Digital Forensic Research Workshop*, pages 11–13, 2004.
18. Harlan Carvey. The windows registry as a forensic resource. *Digital Investigation*, 2(3):201–205, 2005.
19. Harlan Carvey. *Windows Registry Forensics: Advanced Digital Forensic Analysis of the Windows Registry*. Elsevier, 2011.
20. Harlan Carvey. Regripper. Available online on Mar-2016 at <http://www.forensicwiki.org/Regripper>, 2016.
21. Eoghan Casey. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. Academic press, 2011.
22. Yoan Chabot, Aurélie Bertaux, Christophe Nicolle, and M-Tahar Kechadi. A complete formalized knowledge representation model for advanced digital forensics timeline analysis. *Digital Investigation*, 11:S95–S105, 2014.
23. Yoan Chabot, Aurélie Bertaux, Christophe Nicolle, and Tahar Kechadi. An ontology-based approach for the reconstruction and analysis of digital incidents timelines. *Digital Investigation*, 15:83–100, 2015.
24. Belkacem Chikhaoui, Shengrui Wang, and Hélène Pigot. ADR-SPLDA: Activity discovery and recognition by combining sequential patterns and latent dirichlet allocation. *Pervasive and Mobile Computing*, 8(6):845–862, 2012.
25. Alexander Clark. The syntactic concept lattice: another algebraic theory of the context-free languages? *Journal of Logic and Computation*, 25(5):1203–1229, 2013.
26. Diane J Cook, Narayanan C Krishnan, and Parisa Rashidi. Activity discovery and activity recognition: a new partnership. *IEEE Transactions on Cybernetics*, 43(3):820–828, 2013.
27. Sau Dan Lee and Luc De Raedt. *Constraint Based Mining of First Order Sequences in SeqLog*, pages 154–173. Springer Berlin Heidelberg, 2004.

28. Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2004.
29. Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
30. Oxford Dictionaries. Abstraction, 2016. Available online on Apr-2016 at http://www.oxforddictionaries.com//us//definition//american_english//abstraction.
31. Dario V Forte. The PTK: an alternative advanced interface for Sleuth Kit. *Network Security*, 2008(4):10–13, 2008.
32. Simson L Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 4:2–12, 2007.
33. Simson L Garfinkel. Automating disk forensic processing with SleuthKit, XML and Python. In *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE'09. Fourth International IEEE Workshop on*, pages 73–84. IEEE, 2009.
34. Simson L Garfinkel. Digital forensics research: the next 10 years. *Digital Investigation*, 7:S64–S73, 2010.
35. Pavel Gladyshev and Ahmed Patel. Finite state machine approach to digital event reconstruction. *Digital Investigation*, 1(2):130–149, 2004.
36. Ashvin Goel, Wu-chang Feng, David Maier, and Jonathan Walpole. Forensix: a robust, high performance reconstruction system. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 155–162. IEEE, 2005.
37. Greg Gogolin. The digital crime tsunami. *Digital investigation*, 7(1):3–8, 2010.
38. Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.
39. Tao Gu, Zhanqing Wu, Xianping Tao, Hung Keng Pung, and Jian Lu. EP-SICAR: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition. In *IEEE International Conference on Pervasive Computing and Communications, 2009. PerCom 2009*, pages 1–9. IEEE, 2009.
40. K Gudjonsson. Mastering the super timeline with log2timeline. *SANS Institute*, 2010.
41. J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

42. Christopher Hargreaves and Howard Chivers. Recovery of encryption keys from memory using a linear scan. In *Third International Conference on Availability, Reliability and Security, 2008. ARES 08.*, pages 1369–1376. IEEE, 2008.
43. Christopher Hargreaves and Jonathan Patterson. An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigations*, 9:S69–S79, 2012.
44. Sara V Hart, John Ashcroft, and Deborah J Daniels. Forensic examination of digital evidence: a guide for law enforcement. *National Institute of Justice NIJ-US, Washington DC, USA, Tech. Rep. NCJ*, 199408, 2004.
45. Rim Helaoui, Mathias Niepert, and Heiner Stuckenschmidt. Recognizing interleaved and concurrent activities: a statistical-relational approach. In *2011 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9. IEEE, 2011.
46. John Hopcroft and Jeffrey Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Longman Publishing, 1969.
47. Derek Hao Hu and Qiang Yang. Cigar: concurrent and interleaving goal and activity recognition. In *Proceedings of the 2008 Association for the Advancement of Artificial Intelligence Conference*, volume 8, pages 1363–1368, 2008.
48. Ewa Huebner, Derek Bem, and Cheong Kai Wee. Data hiding in the NTFS file system. *Digital Investigation*, 3(4):211–226, 2006.
49. D.A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):275–303, 1954.
50. Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.
51. Joshua James, Pavel Gladyshev, Mohd Taufik Abdullah, and Yuandong Zhu. Analysis of evidence using formal event reconstruction. In *Digital Forensics and Cyber Crime*, pages 85–98. Springer, 2009.
52. Joshua I James and Pavel Gladyshev. Automated inference of past action instances in digital investigations. *International Journal of Information Security*, 14(3):249–261, 2015.
53. Sven Kälber, Andreas Dewald, and Steffen Idler. Forensic zero-knowledge event reconstruction on filesystem metadata. In *Sicherheit*, pages 331–343. Citeseer, 2014.
54. Muhammad Naeem Ahmed Khan. Performance analysis of bayesian networks and neural networks in classification of file system activities. *Computers & Security*, 31(4):391–401, 2012.

55. Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the Nineteenth ACM Symposium on Operating System Principles*, pages 223–236. ACM, 2003.
56. Laks VS Lakshmanan, Raymond Ng, Jiawei Han, and Alex Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *ACM SIGMOD Record*, volume 28, pages 157–168. ACM, 1999.
57. Kevin Lang. Random DFA’s can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 45–62. ACM, 1992.
58. Kevin Lang, Barak Pearlmutter, and Rodney Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, pages 1–12. Springer, 1998.
59. Andreas D Lattner and Otthein Herzog. Constraining the search space in temporal pattern mining. In *LWA*, pages 314–321, 2006.
60. V.I. Levenshtein. Binary codes capable of correct deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
61. Yi-Ching Liao and Hanno Langweng. Resource-based event reconstruction of digital crime scenes. In *IEEE Joint Intelligence and Security Informatics Conference*, pages 129–136. IEEE, 2014.
62. Eugene Libster and Jesse D Kornblum. A proposal for an integrated memory acquisition mechanism. *ACM SIGOPS Operating Systems Review*, 42(3):14–20, 2008.
63. David Lillis, Brett Becker, Tadhg O’Sullivan, and Mark Scanlon. Current challenges and future research areas for digital forensic investigation. *arXiv preprint arXiv:1604.03850*, 2016.
64. Nizar R Mabroukeh and Christie I Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR)*, 43(1):3, 2010.
65. Mandiant. Memoryze. Available online on Aug2009 at [http://www.mandiant.com /software/memoryze.htm](http://www.mandiant.com/software/memoryze.htm).
66. Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovering frequent episodes in sequences extended abstract. In *1st Conference on Knowledge Discovery and Data Mining*, 1995.
67. Albert Marcella and Doug Menendez. *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidency of Computer Crimes, Second Edition*. Auerbach Publications, 2007.

68. Microsoft. What information appears in event logs. *Available online on Mar-2016 at <https://msdn.microsoft.com/en-us/library/windows/what-information-event-logs-event-viewer#1TC=windows-7>*, 2016.
69. Microsoft. Windows event log tools. *Available online on Mar-2016 at [https://msdn.microsoft.com/en-us/library/windows/desktop/dd996929\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dd996929(v=vs.85).aspx)*, 2016.
70. Microsoft. Log parser 2.2. *Available online on July-2017 at <https://technet.microsoft.com/en-us/scriptcenter/dd919274.aspx>*, 2017.
71. Joseph Modayil, Tongxin Bai, and Henry Kautz. Improving the recognition of interleaved activities. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 40–43. ACM, 2008.
72. Carl H Mooney and John F Roddick. Sequential pattern mining—approaches and algorithms. *ACM Computing Surveys (CSUR)*, 45(2):19, 2013.
73. Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys (CSUR)*, 33(1):31–88, 2001.
74. Junghoon Oh, Seungbong Lee, and Sangjin Lee. Advanced evidence collection and analysis of web browser activity. *Digital Investigation*, 8:S62–S70, 2011.
75. James Okolica and Gilbert Peterson. A compiled memory analysis tool. In Kam-Pui Chow and Sujeet Sheno, editors, *Advances in Digital Forensics VI*, pages 195–204. Springer, 2008.
76. Mark Olsen, Russell Horton, and Glenn Roe. Something borrowed: sequence alignment and the identification of similar passages in large text collections. *Digital Studies/Le Champ Numérique*, 2(1), 2011.
77. J. Olson and M. Boldt. Computer forensic timeline visualization tool. *Digital Investigations*, 6(S1), 2009.
78. Richard E Overill, Jantje AM Silomon, and Keith A Roscoe. Triage template pipelines in digital forensic investigations. *Digital Investigation*, 10(2):168–174, 2013.
79. Donald J Patterson, Dieter Fox, Henry Kautz, and Matthai Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Proceedings of the Ninth IEEE International Symposium on Wearable Computers, 2005*, pages 44–51. IEEE, 2005.
80. Mark Pollitt. History, historiography and the hermeneutics of the hard drive. In *Advances in Digital Forensics IX*, pages 3–17. Springer, 2013.

81. Darren Quick and Kim-Kwang Raymond Choo. Data reduction and data mining framework for digital forensic evidence: storage, intelligence, review and archive. *Trends & Issues in Crime and Criminal Justice*, 480:1–11, 2014.
82. Darren Quick and Kim-Kwang Raymond Choo. Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294, 2014.
83. Sriram Raghavan. Digital forensic research: current state of the art. *CSI Transactions on Information and Communication Technology*, 1(1):91–114, 2013.
84. Parisa Rashidi, Diane J Cook, Lawrence B Holder, and Maureen Schmitter-Edgecombe. Discovering activities to recognize and track in a smart environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):527–539, 2011.
85. Rekall. Rekall forensics. Available online on Apr-2016 at <http://www.rekall-forensic.com/about.html>.
86. Golden G Richard III and Vassil Roussev. Next-generation digital forensics. *Communications of the ACM*, 49(2):76–80, 2006.
87. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.
88. Martin Rohrmeier, Qiufang Fu, and Zoltan Dienes. Implicit learning of recursive context-free grammars. *PloS one*, 7(10):e45885, 2012.
89. Vassil Roussev, Candice Quates, and Robert Martell. Real-time digital forensics and triage. *Digital Investigation*, 10(2):158–167, 2013.
90. Duane M Rumbaugh. *Language learning by a chimpanzee: the Lana project*. Academic Press, 2014.
91. Marja Ruotsalainen, Timo Ala-Kleemola, and Ari Visa. GAIS: A method for detecting interleaved sequential patterns from imperfect data. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining*, pages 530–534. IEEE, 2007.
92. Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.
93. Markus G.R. Sause, Alexander Gribov, Anthony R. Unwin, and Siegfried Horn. Pattern recognition approach to identify natural clusters of acoustic emission signals. *Pattern Recognition Letters*, 33(1):17–23, 2012.

94. Adrian Shaw and Alan Browne. A practical and robust approach to coping with large volumes of data submitted for digital forensic examination. *Digital Investigation*, 10(2):116–128, 2013.
95. SQLite. Sqlite, 2017.
96. Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. In *Grammatical Inference and Applications*, pages 106–118. Springer, 1994.
97. Lawrence Stone. The revival of narrative: reflections on a new old history. *Past & Present*, 85:3–24, 1979.
98. Thomas Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley, 1988.
99. M. Suiche. Win32dd. Available online on Aug-2009 at <http://win32dd.msliche.net/>.
100. Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. Efficient single-pass frequent pattern mining using a prefix-tree. *Information Sciences*, 179(5):559–583, 2009.
101. Joshua M Temkin and Mark R Gilder. Extraction of protein interaction information from unstructured text using a context-free grammar. *Bioinformatics*, 19(16):2046–2053, 2003.
102. B. Trakhenbrot and Y. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company, 1973.
103. Benjamin Turnbull and Suneel Randhawa. Automated event and social network extraction from digital evidence sources with ontological mapping. *Digital Investigation*, 13:94–106, 2015.
104. Benjamin Turnbull, Robert Taylor Sr, and Barry Blundell Sr. The anatomy of electronic evidence—quantitative analysis of police e-crime data. In *Proceedings of the 2009 International Conference on Availability, Reliability and Security*, pages 143–149. IEEE, 2009.
105. Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.
106. Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, pages 1–43, 2015.
107. Walmart hard drives, 2016. Available online on May-2016 at www.walmart.com/search/?query=hard%20drives.

108. Aaron Walters and Nick Petroni. Volatools: Integrating volatile memory forensics into the digital investigation process, 2007. Black Hat DC 2007.
109. Daniel White and Joachim Metz. Plaso - super timeline of all things, 2017.
110. Tengke Xiong, Shengrui Wang, Qingshan Jiang, and Joshua Zhexue Huang. A novel variable-order markov model for clustering categorical sequences. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2339–2353, 2014.
111. Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
112. Chung-Huang Yang and Pei-Hua Yen. Fast deployment of computer forensics with USBs. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA)*, pages 413–416. IEEE, 2010.
113. Jiong Yang and Wei Wang. Towards automatic clustering of protein sequences. In *Proceedings of the IEEE Computer Society Bioinformatics Conference*, pages 175–186. IEEE, 2002.
114. Jiong Yang and Wei Wang. Cluseq: Efficient and effective sequence clustering. In *IEEE Proceedings of the 19th International Conference on Data Engineering*, pages 101–112. IEEE, 2003.
115. Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 21-12-2017			2. REPORT TYPE Dissertation		3. DATES COVERED (From — To) Sept 2008 — Dec 2017	
4. TITLE AND SUBTITLE TEMPORAL EVENT ABSTRACTION AND RECONSTRUCTION					5a. CONTRACT NUMBER	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Okolica, James S., Mr.					5d. PROJECT NUMBER	
					5e. TASK NUMBER	
					5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-DS-17-D-004	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT Digital forensic examiners are being overwhelmed by the increasing demand for their services. This research applies sequence pattern mining (SPM) to digital forensics and develops a technique that uses the sequences to generates rules that summarize digital artifacts into human understandable activities. It extends an existing SPM algorithm, Discontinuous Varied Order Sequence Mining (DVSM), and creates two new SPM algorithms, Single Object Sequence and Loop Abstraction (SOSLA) and Sequence Mining of Temporal Clusters (SMTc). These algorithms extend SPM by attaching attribute variables to items and by developing a new method for addressing interleaving. When these algorithms are applied to constructed use cases, results show a 96% reduction in transactions to review, 100% detection of true positives and no more than a 0.03% detection of false positives.						
15. SUBJECT TERMS Digital Forensics, Sequence Pattern Mining, Temporal Events, Abstraction						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson, AFIT/ENG	
U	U	U	UU	594	19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x4281; gilbert.peterson@afit.edu	