TZWorks® Jump List Parser (*jmp*) Users Guide

Abstract

jmp is a standalone, command-line tool used to extract *SHLLINK* artifacts from Windows files that generate *Jump Lists*. *Jump Lists* became available with Windows 7 and provide a wealth of artifacts for the digital forensic analyst. ***jmp*** can operate on a single file or a collection of files. All artifacts can be outputted in one of three parsable formats for easy inclusion with other forensics artifacts. ***jmp*** runs on Windows, Linux and Mac OS-X.

www.tzworks.com

Contact Info: info@tzworks.com

*Document applies to v0.64 of **jmp***

Updated: Apr 25, 2025

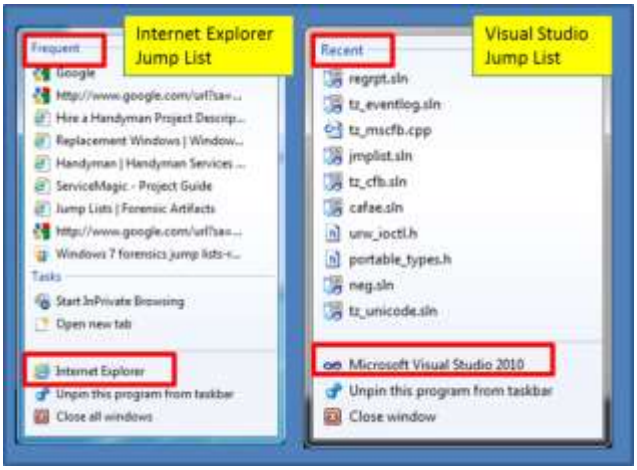
Table of Contents

TZWorks® Jump List Parser (jmp) Users Guide

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=20
Contact Information: info@tzworks.com

1 Introduction

jmp is a command line Windows parser that operates on files that are used to generate *Jump Lists*. *Jump Lists* are a new feature, starting with Windows 7. They are similar to *shortcuts* in that they take one directly to the files or directories that are used on a regular basis. They are different than the normal *shortcut* in that they are more extensible in what information they display. For example, Internet Explorer will use *Jump Lists* to display websites *frequently* visited; Microsoft Office products like Excel, PowerPoint and Word, on the other hand, will show most *recently* opened documents. Below are two examples: (a) one for Most Frequently Used (MFU) list and (b) one for Most Recently Used (MRU) list.



From a user's standpoint, *Jump Lists* increase one's productivity by providing quick access to the files and tasks associated with one's applications. From a forensics standpoint, *Jump Lists* are a good indicator of which files were recently opened or which websites were visited frequently.

The Windows operating system derives the *Jump List* content from two sets of *Destination* files: (a) the first set is the *automaticDestinations* type files. These are the ones the operating system creates and maintains (hence the prefix of 'automatic'). They store information about data file usage. Items are sorted either by Most Recently Used (MRU) or by Most Frequently Used (MFU), depending on the

application. (b) The second set is the *customDestinations* type file. The content contained within, and the tasks specified by this category of file, are maintained by the specific application responsible for that specific *Destination* file. Suffice to say, both formats rely on the *SHLLINK* structure to store much of their information.

Automatic and custom destination files are located in the following areas on the file system, respectively.

a. %APPDATA%\Microsoft\Windows\Recent\AutomaticDestinations\[*AppID*].automaticDestinations-ms
b. %APPDATA%\Microsoft\Windows\Recent\CustomDestinations\[*AppID*].customDestinations-ms

The variable named *%APPDATA%* is used to resolve to the: *C:\Users\<user account>\AppData\Roaming* location. From the path, one can see that each *user account* (or profile) has its own set of *Destination* files.

Per reference [2], the *AppID* (or application identifier) is used as part of the name in the construction of the *Destination* filename. This is a 16 character name in the form of an 8 byte hexadecimal number, and is usually based on the process name, but ultimately can be specified by the application. According to reference [2], the same process should have the same *AppID* across different computer systems and depends on whether the command arguments are the same as well. Reference [3] is a list of various application identifiers that the forensics community has compiled. It is updated as new ones are discovered.

2 Jump List Format/Internals

This is only a brief discussion about the internals of the files that make up the *Jump List*. It is meant to only guide the reader to the current, publically available sources of information, rather than to repeat (and/or explain) the Microsoft specifications.

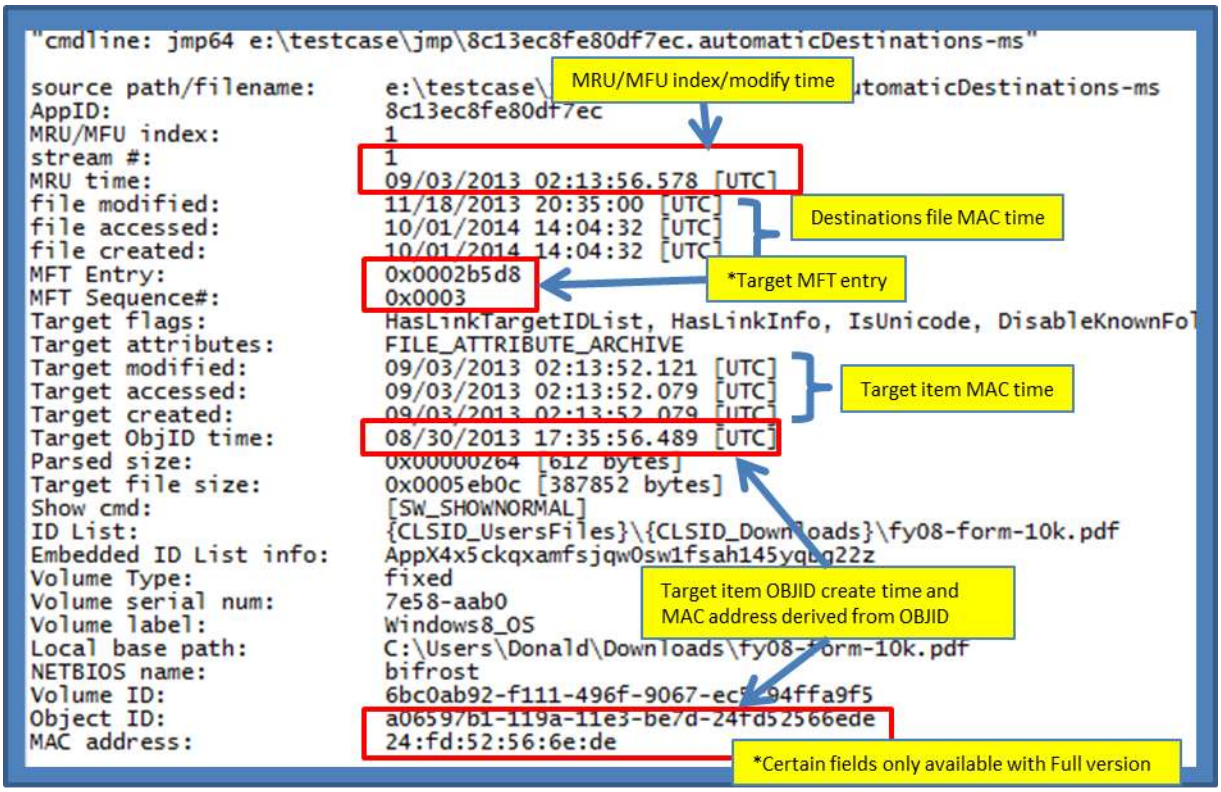
2.1 Automatic Destinations File

Per Microsoft (ref [2]), the *automaticDestinations* file consists of a *Compound File Binary* (CFB) structure as the container for the individual elements. This includes a separate stream for a *DestList* and individual *SHLLINK* streams. As background, the *CFB* architecture has been around for many years and acts like a file-system within a file. This allows each *SHLLINK* structure to be referenced individually via its own separate stream. Both the *CFB* and *SHLLINK* specifications are individually published by Microsoft as part of their open specifications agreement. (ref: MS-CFB [6] and MS-SHLLINK [7], respectively).

The *DestList* stream is a collection of MRU/MFU entries whose format is still being investigated, but contains a timestamp, MRU/MFU entry, corresponding *SHLLINK* stream, NETBIOS name, target file name, object/volume identifiers, and some other unknown information. Combining the *DestList* information with that of the *SHLLINK* information, one can have a useful set of forensic artifacts when determining activity on a computer. Some of these artifacts include:

- Relative frequency of folder or website usage or the last file used for a specific application.
- The path to the target file/directory it references along with modify/access/create timestamps
- The size of the target when it was last accessed.
- Serial number of the volume where the target was stored.
- Network volume share name (if applicable).
- Target attributes, such as whether it was 'read only', 'hidden', 'system', etc.
- One of the MAC addresses associated with the host computer (available when an Object ID is present).

Much of this data can be seen from looking at the default output (or long form) of the ***jmp*** tool (see below).



In certain cases, especially with portable devices, some of the more common data may not be present, such as the target's MACB times or MFT entry and sequence number. Fortunately, there is other useful data that can be extracted from the *IDList* structures. In this example, the device vendor and product identifier can be extracted as well as the serial number of the device and the user's security identifier. Below is an example of what a cell phone may look like after connecting to a computer and accessing files on the portable device.

2.2 Custom Destination File

The *customDestinations* file uses a container structure that is quite different, but simpler, than that of the *automaticDestinations* file. Aside from the initial header at the beginning of the file, the *SHLLINK* structures do not follow the orderly grouping (using streams) like that of the *Compound File Binary* used in the *automaticDestinations* file. Instead, the *SHLLINK* structures are packed sequentially.

Secondly, additional custom metadata can be inserted into *customDestinations* types of files. The contents of this custom data are controlled by the application logging the data. ***jmp***, however, only parses the *SHLLINK* type metadata from these types of files and does not try to parse any of the unique metadata that may have been placed there by the application.

2.3 Pulling out Metadata from the Segmented ItemIDs

Windows uses the Shell *ItemID* to build the path of the file specified for the link. Each ItemID can contain other information beside the segment of the path. This other information can include: (a) MAC times, MFT entry of the segment, and MFT sequence number. To pull out this additional metadata, use the ***-idltimes*** switch. Below is an example of doing this. The additional data outputted is highlighted below.

```
source path/filename:     e:\testcase\jmp\fb3b0dbfee58fac8.automaticDestinations-ms
AppID:                    fb3b0dbfee58fac8
MRU/MFU index:            13
stream #:                 37
MRU time:                 08/01/2013 19:21:18.540 [UTC]
file modified:            11/18/2013 20:26:01 [UTC]
file accessed:            10/01/2014 14:04:33 [UTC]
file created:             10/01/2014 14:04:33 [UTC]
MFT Entry:                0x0000bce9
MFT Sequence#:            0x0009
Target flags:             HasLinkTargetIDList, HasLinkInfo, IsUnicode, DisableKnownFolderA
Target attributes:        FILE_ATTRIBUTE_ARCHIVE
Target modified:          08/12/2013 03:39:23.540 [UTC]
Target accessed:          08/08/2013 19:18:11.402 [UTC]
Target created:           08/08/2013 19:18:11.402 [UTC]
Target ObjID time:        10/21/2013 16:14:28.758 [UTC]
Parsed size:              0x00000422 [1058 bytes]
Target file size:         0x00059625 [366117 bytes]
Show cmd:                 [SW_SHOWNORMAL]
ID List:                  {CLSID_UsersFiles}\Dropbox\Shared Documents\Confidential Analysi
Embedded ID List info:    Word.Document.12
Volume Type:              fixed
Volume serial num:        7e58-aab0
Volume label:             Windows8_OS
Network name:             \\BIFROST\Users
Local base path:          C:\Users\
Common path:              Donald\Dropbox\Shared Documents\Confidential Analysis Data\Blue
NETBIOS name:             bifrost
Volume ID:                6bc0ab92-f111-496f-9067-ec5c94ffa9f5
Object ID:                dc8ffc9d-3a6b-11e3-be8a-24fd52566ede
MAC address:              24:fd:52:56:6e:de
IDList subpath breakout
 segment: {CLSID_UsersFiles}
 segment: Dropbox
  modify [UTC]: 08/08/2013 19:17:00
  access [UTC]: 08/08/2013 19:17:00
  create [UTC]: 08/08/2013 19:17:00
  mft entry#:   0x0000bc5c [48220]
  mft seq#:     0x0004 [4]
 segment: Shared Documents
  modify [UTC]: 08/08/2013 19:18:12
  access [UTC]: 08/08/2013 19:18:12
  create [UTC]: 08/08/2013 19:18:12
  mft entry#:   0x0000bce6 [48358]
  mft seq#:     0x0004 [4]
 segment: Confidential Analysis Data
  modify [UTC]: 08/08/2013 19:18:12
  access [UTC]: 08/08/2013 19:18:12
  create [UTC]: 08/08/2013 19:18:12
  mft entry#:   0x0000bce7 [48359]
  mft seq#:     0x0004 [4]
 segment: Blue Harvest Business Plan.docx
  modify [UTC]: 08/08/2013 19:18:12
  access [UTC]: 08/12/2013 03:39:24
  create [UTC]: 08/08/2013 19:18:12
  mft entry#:   0x0000bce9 [48361]
  mft seq#:     0x0009 [9]
```

Additional metadata broken out by *ItemID* segment using ***-idltimes*** option.

*Extra data only available with Full version

2.4 *jmp* tool parsing vice the *lp* tool

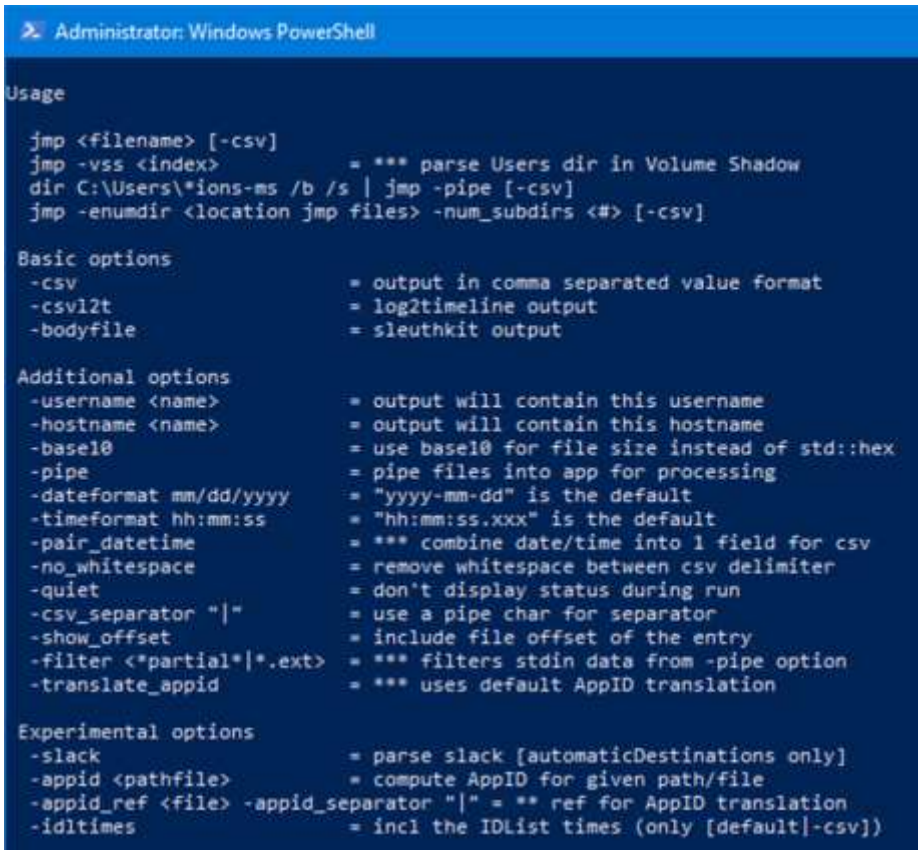
When designing the ***jmp*** tool, the *SHLLINK* parsing engine was taken from the *TZWorks* LNK parsing tool called ***lp*** (ref [5])**.** The ***lp*** engine was wrapped in a compound file stream parsing engine to extract the appropriate streams so the *SHLLINK* structures could be parsed with assured accuracy.

As an aside, the ***lp*** tool also has an option (unlike the ***jmp*** tool) to parse *SHLLINK* structures from *raw* unstructured data. Using this capability, one can pull out *SHLLINK* metadata that is buried within an image of a volume. Furthermore, ***lp*** can also parse *SHLLINK* structures from within a compound file, similar to that of an *automaticDestinations* file. The ***lp*** tool, however, does not associate the *automaticDestinations* file's MRU/MFU data for each *SHLLINK* structure parsed, and hence, the reason that the ***jmp*** tool was created. More information about what the ***lp*** tool can do, please refer to the *TZWorks* website [5].

3 How to Use *jmp*

While the ***jmp*** tool doesn't require one to run with administrator privileges, without doing so will restrict one to only looking at files available to the current logged in account or those common to the operating system. Thus, to access either other accounts, you should run this tool with administrator privileges.

One can display the menu options by typing in the executable name without parameters. A screen shot of the menu is shown below.



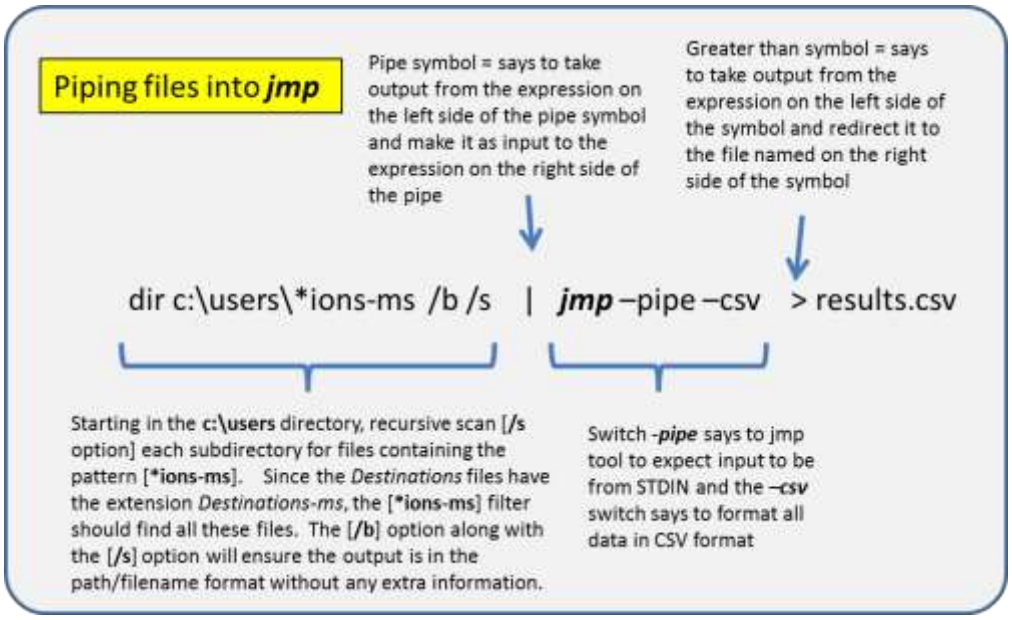
To parse an individual *Destinations* file, use the following notation:

jmp [Destinations filename] > results.txt

Without specifying one of the format options, the output is rendered in a default, unstructured output. The snapshot in the section (2.1) above is an example of what this output looks like. The information is useful if one is not trying to parse any artifacts into a database. Notice that the example above, the output is redirected to a text file called 'results.txt'. Since the output that is generated is usually very long (and wide, if using the CSV option), it is recommended that one redirect the output of the command into a file as show above.

The ***-csv*** (comma separated value) option will render the output so that all the metadata is rendered with one *SHLLINK* record per line, where each field is delimited by a comma. The other two output formats are the: (a) ***-csvl2t*** and (b) ***-bodyfile***. Each will attempt to conform to either the *log2timeline* format or the *SleuthKit's body-file* format, as appropriate.

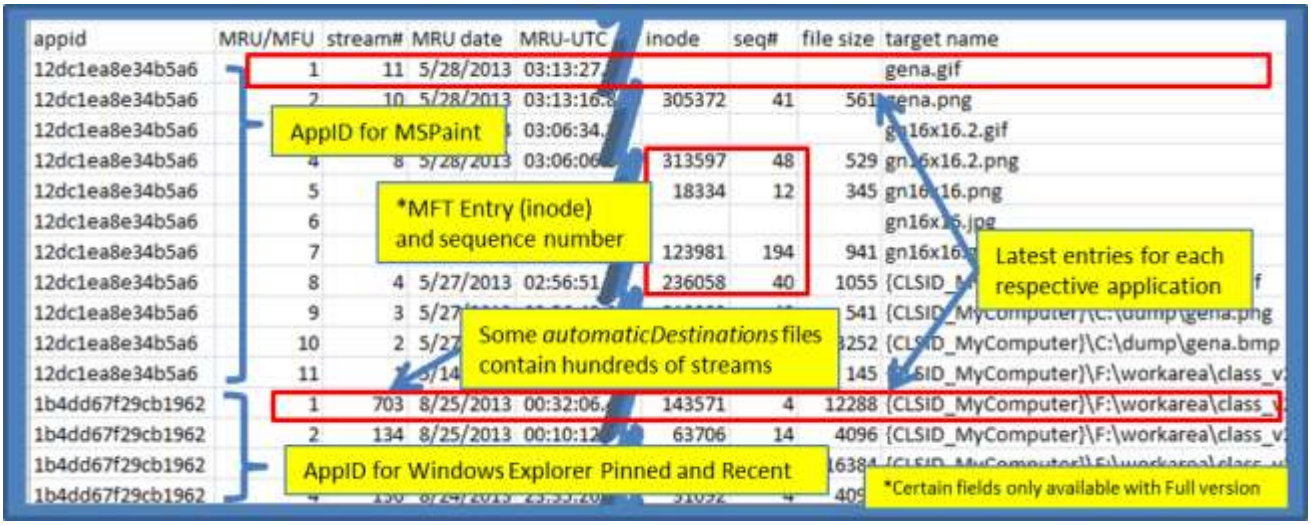
While parsing one *Destinations* file is useful, one will usually want to parse all the *Destinations* files that are on a system or in a set of directories. One way to do this is to pipe in all the paths/filenames of the *Destinations* type files one wishes to parse into ***jmp***. To allow ***jmp*** to receive data from an input pipe, one needs to invoke the ***-pipe*** switch. This will allow ***jmp*** to receive a separate path/filename per line as input. To provide this input, one can use the Windows built-in ***dir*** command along with some of its own switches to get the desired result. For those not familiar with syntax that uses a *pipe* or the ***dir*** command and options, the figure below provides annotations to what each portion in the command is doing.



The above syntax will process all the *Destinations-ms* files that are located anywhere in the c:\users directory or subdirectories. This assumes, of course, that one is running with administrator privileges.

If one cannot use the ***-pipe*** option, one can use the experimental ***-enumdir*** option, which has similar functionality with more control. The ***-enumdir*** option takes as its parameter the folder to start with. It also allows one to specify the number of subdirectories to evaluate using the ***-num_subdirs <#>*** sub-option.

When analyzing the CSV output the *automaticDestinations* metadata will include MRU/MFU index, stream and timestamp. The annotated snapshot below has been trimmed to show this information.



On the left of the output, one can see the application identifier. Matching this identifier to a known application can be done by visiting the http://www.forensicswiki.org/wiki (see ref. [3]), where there are tables of application IDs matched to their respective application.

The MRU/MFU index is a chronological list of entries, where #1 is the latest. Each application ID has each own sequential list. Following the index value is the stream number (or directory name) that holds the data in the compound file. It is not uncommon to find hundreds of streams, where each stream (with a couple exceptions) contains a *SHLLINK* data structure. The MRU/MFU entry has a unique timestamp that shows when that entry was updated. Finally, some of the entries contain the target MFT entry (or *inode*) and sequence number. Combining the MRU/MFU data with that of the *SHLLINK* metadata and the MFT entry provides a wealth of forensic information to the investigator.

3.1 Translating the Application Identifier to the Application Name

An application identifier (*AppID*) can be set by the application itself or it can be generated by the operating system using a set of rules. The ***jmp*** tool has two options to assist in this area.

3.1.1 Going from path/name to AppID

The first option is to go from path/filename to *AppID*, using the ***-appid <pathfile>*** option. The algorithm first converts the path to uppercase Unicode and then substitutes any known Windows folders with the associated GUID identifier. Based on the string final string generate, the CRC is computed and result is a possible AppID. Since the *AppID* is very sensitive to the inputted path, one needs to keep this in mind when trying to compute *AppID's* from an executables path and name. A good example of this is trying to compute the *AppID* for the notepad application. If one opens a command prompt and issues the command: ***where notepad***, one may see two possible paths as I do on

my default Win7 box. One exists at the path C:\windows\System32\. Since this is the first one shown, it is considered the default one. But there is one also at the path c:\windows\. To complicate matters, the Win7 box happens to be a 64 bit OS, so there happens to be a 3rd path at the c:\windows\SysWOW64\ directory. Out of these 3 possible paths for notepad, the *AppID* could be using the GUID version of the path instead of the ones listed before. The Windows OS likes to use GUID identifiers for common folders. Given this, we now have 6 possible *AppID's* for notepad. The *AppID's* in the output below are called hashes.

```
> where notepad

C:\Windows\System32\notepad.exe
C:\Windows\notepad.exe

> jmp64 -appid c:\windows\notepad.exe

hash: 47592b67dd97a119 : {F38BF404-1D43-42F2-9305-67DE0B28FC23}\NOTEPAD.EXE
hash: 33b1533ff9c7e8af : C:\WINDOWS\NOTEPAD.EXE

> jmp64 -appid c:\windows\system32\notepad.exe

hash: 9b9cdc69c1c24e2b : {1AC14E77-02E7-4E5D-B744-2EB1AE5198B7}\NOTEPAD.EXE
hash: aa9870e8e6c0d630 : C:\WINDOWS\SYSTEM32\NOTEPAD.EXE

> jmp64 -appid c:\windows\SysWOW64\notepad.exe

hash: 918e0ecb43d17e23 : {D65231B0-B2F1-4857-A4CE-A8E7C6EA7D27}\NOTEPAD.EXE
hash: c3e94993c9780ce0 : C:\WINDOWS\SYSWOW64\NOTEPAD.EXE
```

One should note that the computation of the *AppID* did not care whether the application was 32 bit or 64 bit, but just the path it came from. This means that for a 32 bit OS, the *AppID* for C:\Windows\System32\notepad.exe would be identical to the *AppID* for a 64 bit OS with the same path. So while some references show the mapping between the 32 and 64 bit in the appid.txt file, it could be misleading.

3.1.2 Going from AppID to Application Name

The second option takes a file as its argument that already has the mapping of *AppID's* to application names in a file. This option uses the ***-appid_ref <file>*** to tell ***jmp*** to use these associations to output the mapped application name in the output.

Included in the distribution of ***jmp*** is a pre-generated file (named appids.txt) that contain the mappings included from the online forensics-wiki website (see reference 3 at the end of this document). One is cautioned and should verify the accuracy of the data in this file, since we just took the output from the website and formatted it in a way that the ***jmp*** tool could easily parse it and use the mappings.

If one desires to add new entries to this file, then use the following rules:

- Blank lines are ignored
- Lines starting with a forward double slash are ignored and used for comments

- An AppID mapping to an application name is separated by a pipe character. If we took the AppIDs computed for notepad above, one could generate a few entries this way:

```
// appid's for notepad
9b9cdc69c1c24e2b | notepad.exe (system32)
918e0ecb43d17e23 | notepad.exe (SysWOW64)
47592b67dd97a119 | notepad.exe (windows)
```

3.2 Handling Volume Shadow Copies

For starters, to access Volume Shadow copies, one needs to be running with administrator privileges. Also, Volume Shadow copies, as is discussed here, only apply to Vista, Win7, Win8 and beyond. It does not apply to Windows XP.

To tell ***jmp*** to look at a Volume Shadow, one needs to use the ***-vss <index of volume shadow>*** option. This then points ***jmp*** at the appropriate Volume Shadow and it starts processing the various user *automatic* and *custom* destination files.

In addition to the ***-vss <index of volume shadow>***, we've built in a shortcut syntax to access a specific file in a specified Volume Shadow copy, via the ***%vss%*** keyword. This internally gets expanded into *\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy*. Thus, to access index 1 of the volume shadow copy, one would prepend the keyword and index, like so, ***%vss%1*** to the normal path of the file. For example, to access a file located in the *testuser* account from the HarddiskVolumeShadowCopy1, the following syntax can be used:

```
jmp %vss%1\Users\testuser\AppData\Roaming\Microsoft\Windows\Recent\
AutomaticDestinations\1b4dd67f29cb1962.automaticDestinations-ms  > out.txt
```

To determine which indexes are available from the various Volume Shadows, one can use the Windows built-in utility ***vssadmin***, as follows:

```
vssadmin list shadows
```

To filter some of the extraneous detail, type

```
vssadmin list shadows | find /i "volume"
```

While the amount of data can be voluminous from that above command, the keywords one needs to look for are names that look like this:

```
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
```

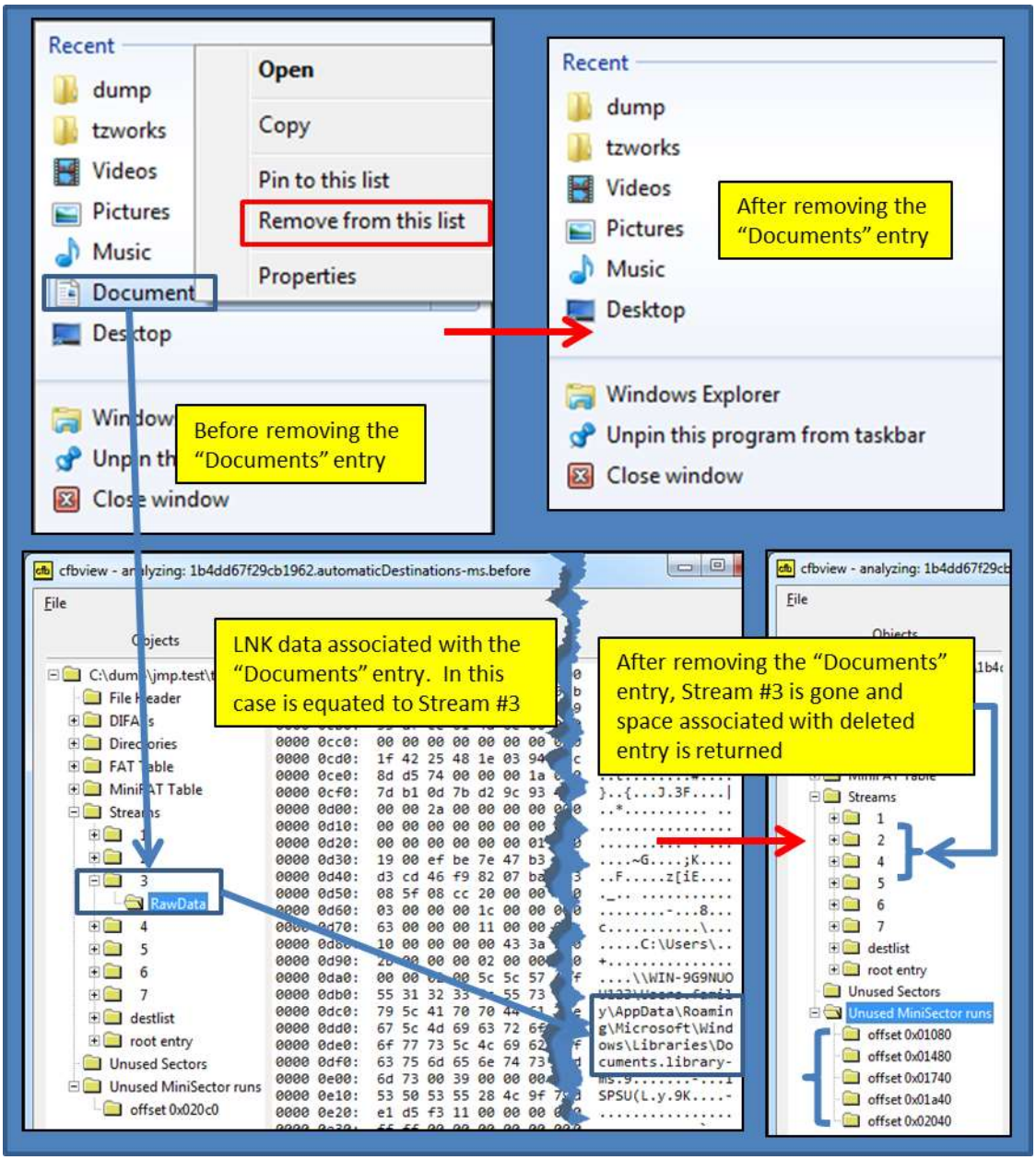
...

From the above, notice the number after the word *HarddiskvolumeShadowCopy*. It is this number that is passed as an argument to the ***-vss*** option.

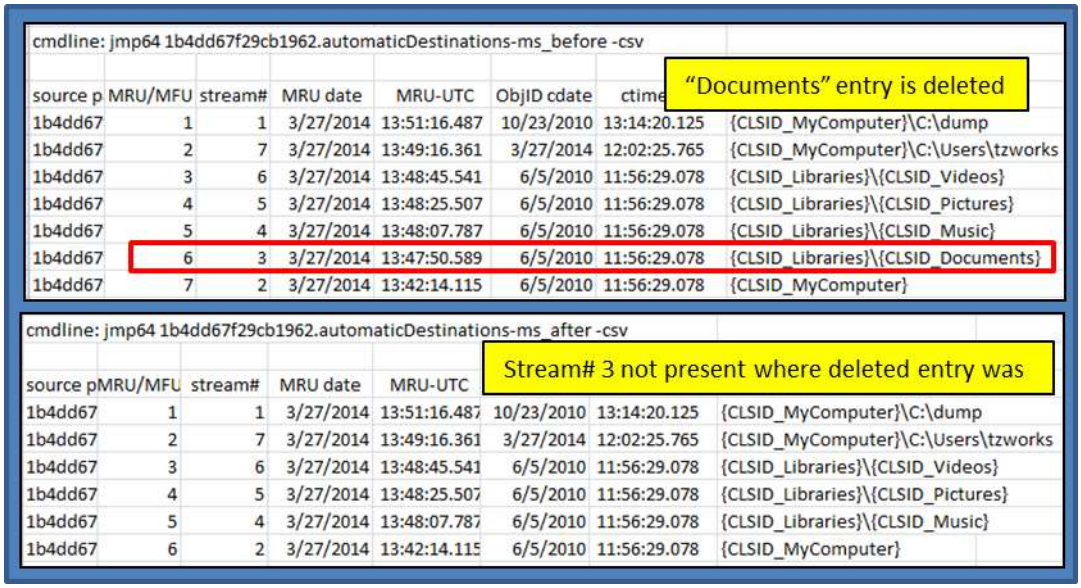
3.3 Understanding Deleted JumpList Entries

As discussed earlier, the *automaticDestinations* structure is like a FAT file system within a file. Therefore, it stands to reason that deleted entries will be persistent until the Jump file shrinks or the delete space is reused.

Below is a series of annotated screenshots of a Jump List from Windows Explorer. The images show what happens when an entry is deleted from the list and how it affects the metadata in the Compound File Binary (CFB) data structure used to encapsulate automaticDestinations files. For this example, the "Documents" entry is deleted. From the bottom left image, Stream #3 is highlighted to show what the data looks like prior to the deletion. On the bottom right image, one can see that Stream #3 is now absent from the stream list. Also highlighted is the additional number of mini-sector chunks that are now available for use by the CFB container. The tool being used for this analysis is a *CFBViewer* that we wrote for internal use at *TZWorks* to help analyze the internals of CFB files.



When looking at the results from ***jmp***, one will see the following data before and after the deletion of the entry. The snapshot below collapses many of the fields to show both the stream# and entry name for clarity.



cmdline: jmp64 1b4dd67f29cb1962.automaticDestinations-ms_before -csv

"Documents" entry is deleted

source p	MRU/MFU	stream#	MRU date	MRU-UTC	ObjID cdate	ctime	
1b4dd67	1	1	3/27/2014	13:51:16.487	10/23/2010	13:14:20.125	{CLSID_MyComputer}\C:\dump
1b4dd67	2	7	3/27/2014	13:49:16.361	3/27/2014	12:02:25.765	{CLSID_MyComputer}\C:\Users\tzworks
1b4dd67	3	6	3/27/2014	13:48:45.541	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Videos}
1b4dd67	4	5	3/27/2014	13:48:25.507	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Pictures}
1b4dd67	5	4	3/27/2014	13:48:07.787	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Music}
1b4dd67	6	3	3/27/2014	13:47:50.589	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Documents}
1b4dd67	7	2	3/27/2014	13:42:14.115	6/5/2010	11:56:29.078	{CLSID_MyComputer}

cmdline: jmp64 1b4dd67f29cb1962.automaticDestinations-ms_after -csv

Stream# 3 not present where deleted entry was

source p	MRU/MFU	stream#	MRU date	MRU-UTC			
1b4dd67	1	1	3/27/2014	13:51:16.487	10/23/2010	13:14:20.125	{CLSID_MyComputer}\C:\dump
1b4dd67	2	7	3/27/2014	13:49:16.361	3/27/2014	12:02:25.765	{CLSID_MyComputer}\C:\Users\tzworks
1b4dd67	3	6	3/27/2014	13:48:45.541	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Videos}
1b4dd67	4	5	3/27/2014	13:48:25.507	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Pictures}
1b4dd67	5	4	3/27/2014	13:48:07.787	6/5/2010	11:56:29.078	{CLSID_Libraries}\{CLSID_Music}
1b4dd67	6	2	3/27/2014	13:42:14.115	6/5/2010	11:56:29.078	{CLSID_MyComputer}

3.4 Parsing Slack Entries (Issues and what can be done)

Now that one has a basic understanding on how deleted entries may persist in the CFB Container, one can then target these unused mini-sectors to carve out any deleted LNK files. First of all, the bad news is that there is no way to tell how these newly free mini-sectors were aligned in a run when representing the data in a LNK file. This means if the CFB file was severely fragmented, one will not be able to reconstruct many of the deleted entries. The second point we have observed from empirical data is when a run of mini-sectors frees up, and it happens that these same mini-sectors comprise an entire sector (many mini-sectors make up 1 sector), then the CFB file is shrunk down in size and the sector is no longer backed by file data. These two issues make reconstructing a deleted LNK file a hit and miss type problem.

What we can do is look at the freed up mini-sectors and reconstruct a larger dataset by ordering them from lowest offset to highest offset. From this ordered set of data, we can then scan for LNK signatures, and if found, we can parse the data starting with the signature. With that as the initial conditions, there are two tools that can do this within our toolset. The first is to use a tool such as ***lp*** (ref: https://tzworks.com/prototype_page.php?proto_id=11), which understands how to reconstruct CFB files into a contiguous run of data and then scan the resulting data for LNK signatures. While this works fine for allocated chunks of data, it has the same issues with freed up chunks of data discussed before. The second option is to use the ***jmp -slack*** option to scan these unallocated chunks of data for LNK signatures, and if found, parses them.

Since any results returned from slack are not registered with the *DestList* stream in the CFB container, the MRU/MFU and other metadata associated with the *DestList* will not present with these retrieved

entries. Furthermore, unallocated chunks of data may or may not be aligned in contiguous blocks, and therefore the results returned that are labeled as 'slack' may be corrupted or have errors in the data.

Below is an example of running ***jmp*** on a Jump List with a number of deleted entries, but only one was able to be parsed. The output was trimmed to show just the entries that are changed during a reconstruction of the deleted entry. If using slack results for analysis, one is encouraged to use the ***-show_offset*** option as well. This option will output the actual offset in the automaticDestinations file where the entry was parsed from (keep in mind, while the starting offset is valid, the rest of the data may not be contiguous and could be fragmented). But for the slack data, it will allow one to later go and verify the results manually with any hex editor.

cmdline: jmp64 1b4dd67f29cb1962.automaticDestinations-ms_some_delete -slack -csv -show_offset

source pa	source type	MRU/MFU	stream#	MRU date	MRU-UTC	LNK offset	target name
1b4dd67f2	JMPLIST (automatic)	1	6	3/27/2014	13:48:45.541	0x1800	{CLSID_Libraries}\{CLSID_Videos}
1b4dd67f2	JMPLIST (automatic)	2	5	3/27/2014	13:48:25.507	0x1500	{CLSID_Libraries}\{CLSID_Pictures}
1b4dd67f2	JMPLIST (automatic)-slack	na	na	na	na	0x11c0	{CLSID_Libraries}\{CLSID_Music}

4 Known Issues

jmp doesn't parse some of the fields in the *SHLLINK* structures documented by the Microsoft specification. As time permits, future versions will incorporate incremental capabilities to handle these additional fields.

For CSV (comma separated values) output, there are restrictions in the characters that are outputted. Since commas are used as a separator, any data containing commas are replaced with a space. For the default (non-CSV) output no changes are made to the data. To address this issue, an option was added to change the CSV default separator character from the comma (,) to whatever is desired. The pipe (|) character is a good choice, since it doesn't overlap with characters in filenames. (This option is discussed below, reference ***-csv_separator***).

For Linux and Mac builds, 'file create' date is not shown, but the 'system changed' time is instead.

(Windows only). When processing filenames containing characters that are not ASCII one option is to change the code page of the command window from the default code page to UTF-8. This can be done via the command:

chcp 65001

5 Available Options

Option	Description
-csv	Outputs the data fields delimited by commas. Since filenames can have commas, to ensure the fields are uniquely separated, any commas in the filenames get converted to spaces.
-csvl2t	Outputs the data fields in accordance with the log2timeline format.
-bodyfile	Outputs the data fields in accordance with the 'body-file' version3 specified in the SleuthKit. The date/timestamp outputted to the body-file is in terms of UTC. So if using the body-file in conjunction with the mactime.pl utility, one needs to set the environment variable TZ=UTC.
-csv_separator	Used in conjunction with the ***-csv*** option to change the CSV separator from the default comma to something else. Syntax is ***-csv_separator "|"*** to change the CSV separator to the pipe character. To use the tab as a separator, one can use the ***-csv_separator "tab"*** OR ***-csv_separator "\t"*** options.
-base10	Ensure all size/address output is displayed in base-10 format vice hexadecimal format. Default is hexadecimal format
-username	Option is used to populate the output records with a specified username. The syntax is ***-username <name to use>***.
-hostname	Option is used to populate the output records with a specified hostname. The syntax is ***-hostname <name to use>***.
-pipe	Used to pipe files into the tool via STDIN (standard input). Each file passed in is parsed in sequence.
-enumdir	Experimental. Used to process files within a folder and/or subfolders. Each file is parsed in sequence. The syntax is ***-enumdir <folder> -num_subdirs <#>***.
-filter	Filters data passed in via STDIN via the ***-pipe*** option. The syntax is ***-filter <"*.ext | *partialname* | ...">***. The wildcard character '*' is restricted to either before the name or after the name.
-vss	Experimental. Extract Jump List data from Volume Shadow. The syntax is ***-vss <index number of shadow copy>***. Only applies to Windows Vista, Win7, Win8 and beyond. Does not apply to Windows XP.
-idltimes	Experimental. Shell item identifiers are grouped together to form a path. Each Item ID can have embedded in it an associated MAC timestamps as well as MFT entry number for the segment of the path that creates the final path. Using this option will display any additional metadata associated with each segment (or Item ID) in the list

-no_whitespace	Used in conjunction with ***-csv*** option to remove any whitespace between the field value and the CSV separator.
-slack	Experimental: *automaticDestination*s files retain some of their deleted entries in slack space. The ***-slack*** option traverses this slack space to extract any additional LNK entries. These entries that are retrieved do not have any MRU/MFU data associated with them.
-appid	Experimental: Points to a path/file combination and computes the AppID. The syntax is ***–appid <file>***.
-appid_ref	Experimental: Points to a text file to translate application identifiers to application names. The syntax is ***–appid_ref <file>***. Distribution contains a sample file called ***appids.txt*** and the data is taken from the forensic wiki (ref: http://www.forensicswiki.org/wiki/List_of_Jump_List_IDs). The file uses a pipe delimiter between the application ID and the application name. If a different delimiter is used, one can use the option ***-appid_separator ","*** to tell ***jmp*** to use a different delimiter (in this case a comma) to parse the AppID file.
-dateformat	Output the date using the specified format. Default behavior is *-dateformat "yyyy-mm-dd"*. Using this option allows one to adjust the format to mm/dd/yy, dd/mm/yy, etc. The restriction with this option is the forward slash (/) or dash (-) symbol needs to separate month, day and year and the month is in digit (1-12) form versus abbreviated name form.
-timeformat	Output the time using the specified format. Default behavior is ***-timeformat "hh:mm:ss.xxx"*** One can adjust the format to microseconds, via ***"hh:mm:ss.xxxxxx"*** or nanoseconds, via ***"hh:mm:ss.xxxxxxxxx"***, or no fractional seconds, via ***"hh:mm:ss"***. The restrictions with this option is a colon (:) symbol needs to separate hours, minutes and seconds, a period (.) symbol needs to separate the seconds and fractional seconds, and the repeating symbol 'x' is used to represent number of fractional seconds. (Note: the fractional seconds applies only to those time formats that have the appropriate precision available. The Windows internal filetime has, for example, 100 nsec unit precision available. The DOS time format and the UNIX 'time_t' format, however, have no fractional seconds). Some of the times represented by this tool may use a time format without fractional seconds, and therefore, will not show a greater precision beyond seconds when using this option.
-pair_datetime	Output the date/time as 1 field vice 2 for csv option
-quiet	Used in conjunction with ***-pipe*** option. This option suppresses status output as each file is processed.

-show_offset	Displays starting offset of the file where the parsed artifact starts. This is used primarily for verification purposes when used in conjunction with a hex-editor during the analysis. (Note: the *automaticDestinations* jumpfile does not necessarily have contiguous sectors of data for a parsed artifact. Therefore, if using this option, be aware that the reconstruction of the data depends on following the allocated chain of sectors and not just looking at the starting offset in a hex-editor and assuming the rest of the data follows after the first sector or mini-sector).
-utf8_bom	All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 *byte order mark* to the output using this option.

6 Authentication and the License File

This tool has authentication built into the binary. The primary authentication mechanism is the digital X509 code signing certificate embedded into the binary (Windows and macOS).

The other mechanism is the runtime authentication, which applies to all the versions of the tools (Windows, Linux and macOS). The runtime authentication ensures that the tool has a valid license. The license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license.

6.1 *Limited* versus *Demo* versus *Full* in the tool's Output Banner

The tools from *TZWorks* will output header information about the tool's version and whether it is running in *limited, demo* or *full* mode. This is directly related to what version of a license the tool authenticates with. The *limited* and *demo* keywords indicates some functionality of the tool is not available, and the *full* keyword indicates all the functionality is available. The lacking functionality in the *limited* or *demo* versions may mean one or all of the following: (a) certain options may not be available, (b) certain data may not be outputted in the parsed results, and (c) the license has a finite lifetime before expiring.

7 References

1. Windows 7 Taskbar – Part 1, The Basics. http://blogs.msdn.com/b/yochay/archive/2009/01/06/windows-7-taskbar-part-1-the-basics.aspx, by Yochay Kiriaty, dated 01/05/2009.
2. Windows 7 Jump Lists, windows7forensics-jumplists-rv3-public-110606164708-phpapp01.pptx, Troy Larson PowerPoint charts.

3. Application Identifiers used in Jump Lists, http://www.forensicswiki.org/wiki/List_of_Jump_List_IDs.
4. Discussion of the Jump List structure, http://www.forensicswiki.org/wiki/Jump_Lists
5. http://tzworks.com/prototype_page.php?proto_id=11, lp tool, Windows LNK Parsing Utility, TZWorks, LLC.
6. [MS-CFB]: Compound File Binary Format, 06/10/2011, sourced from Microsoft Corporation.
7. [MS-SHLLINK]: Shell Link (.LNK) Binary File Format, 11/12/2010, sourced from Microsoft Corporation.
8. Jesse Hager "The Windows Shortcut File Format", Available at http://www.i2slab.com/Papers/The_Windows_Shortcut_File_Format.pdf.
9. *SleuthKit* Body-file format, http://wki.sleuthkit.org
10. Log2timeline CSV format, http://log2timeline.net/