

TZWorks® Windows Journal Parser (*jp*) Users Guide



Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v1.50 of *jp*

Updated: Apr 25, 2025

Abstract

jp is a standalone, command-line tool used to extract change log journal entries from Windows NTFS volumes. From this journal, one can extract: (a) time of change to a file/directory and (b) change type (eg. deleted, renamed, size change, etc). *jp* can operate on a live volume, an image of a volume or an extracted change log journal. All artifacts can be outputted in one of four parsable formats for easy inclusion with other forensics artifacts. *jp* runs on Windows, Linux and macOS.

Table of Contents

1	Introduction	2
2	Change Log Journal Format/Internals.....	2
3	How to Use <i>jp</i>	5
3.1	Processing Volume Shadow Copies	6
3.2	Including old entries from Unallocated Clusters	7
3.3	Including old entries from Volume Shadow Clusters and Slack space	7
3.4	Available Output Options.....	7
3.4.1	CSV Output.....	8
3.4.2	Time resolution and Date format	9
4	Change Journal Reason Code.....	9
5	Known Issues.....	11
6	Available Options	11
7	Authentication and the License File.....	13
7.1	<i>Limited</i> versus <i>Demo</i> versus <i>Full</i> in the tool's Output Banner.....	14
8	References	14

TZWorks® ChangeLog Journal Parser (*jp*)

Users Guide

Copyright © TZWorks LLC

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=5

Contact Information: info@tzworks.com

1 Introduction

jp is a command line tool that targets NTFS change log journals. The change journal is a component of NTFS that will, when enabled, record changes made to files. The change journal is located in the file *\$UsnJrnl* within the alternate data stream *\$J*. Each entry is of variable size and its internal structure is documented by Microsoft via the reference [1].

The change journal will record, amongst other things: (a) time of the change, (b) affected file/directory, and (c) change type (eg. delete, rename, size, etc). This metadata is useful tool when looking at a computer forensically.

Microsoft provides tools to look/affect the change journal as well as a published API to programmatically read from/write to the change log. *jp* however, doesn't make use of the Windows API, but does the parsing by traversing the raw structures. This allows *jp* to be compiled for use on other operating systems to parse the change journal as a component in a forensic toolkit.

Currently there are compiled versions for Windows, Linux and macOS.

2 Change Log Journal Format/Internals

Just because one may have an NTFS volume does not mean that a Change Journal exists on that volume. In fact, with Windows XP, the Change Journal was not active by default. An application or a user could enable one, but short of that happening, it would not be present. Starting around with Windows 7, the change log Journal was active on the system drive by default. Other volumes, however, do not necessarily have the change log journal instantiated.

To check whether the NTFS volume you are analyzing has this type of journal, one can use the built-in Microsoft tool *fsutil*. The functionality we are interested in is the **USN** option. The USN abbreviation stands for Update Sequence Number (**USN**) Journal. In Microsoft documentation, the USN Journal is also known as the Change Journal. In this document, both terms will be used interchangeably. When invoking the **USN** option in *fsutil*, the Microsoft utility shows what USN commands are supported (see below):

```
Administrator: Command Prompt
C:\dump>fsutil usn
----- USN Commands Supported -----
createjournal      Create a USN journal
deletejournal      Delete a USN journal
enumdata           Enumerate USN data
queryjournal        Query the USN data for a volume
readdata           Read the USN data for a file
```

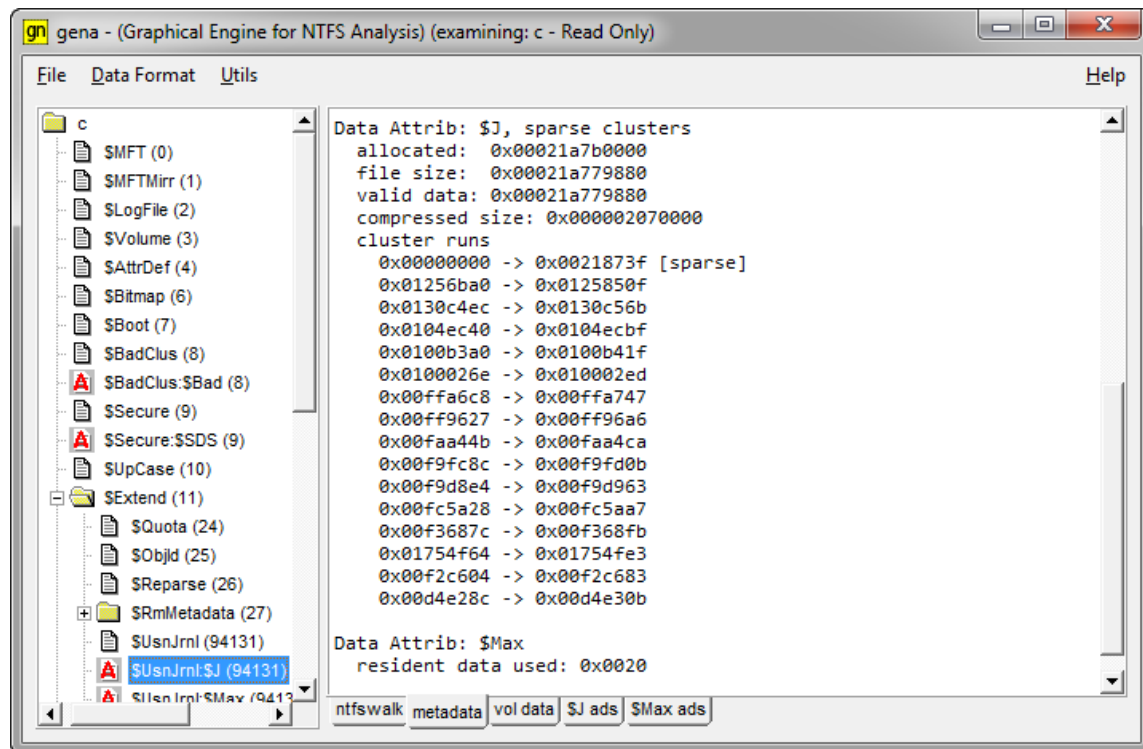
In this case, we are interested in finding out whether a Change Journal is present on a volume we wish to analyze. One does this via: **fsutil usn queryjournal <volume letter>** (see below). If the volume contains a USN Journal, then the statistics regarding the journal are displayed.

```
Administrator: Command Prompt
C:\dump>fsutil usn queryjournal c:
Usn Journal ID      : 0x01cbb59a4696a8bb
First Usn           : 0x00000000279340000
Next Usn            : 0x0000000027b3a1130
Lowest Valid Usn    : 0x0000000000000000
Max Usn             : 0x7fffffffffffffff0000
Maximum Size        : 0x0000000002000000
Allocation Delta    : 0x000000000400000
```

If a USN Journal is not present, the following error message is displayed

```
Administrator: Command Prompt
C:\dump>fsutil usn queryjournal v:
Error: The volume change journal is not active.
```

To find the change journal, one needs to look into the root directory of the volume under analysis. Buried within one of the hidden system files, is an alternate data stream containing the change log journal. Specifically, the journal data is located at the `[root]\$Extend\${UsnJrnl:$J}`, where the `$J` is an alternate data stream. Looking at this location with an NTFS viewer (shown below), one can easily drill down to the proper location and analyze the contents. When examining the cluster run for the `$J` data stream, one will see the beginning clusters are sparse, meaning they are not backed by physical disk clusters, while the later clusters are not sparse (meaning they have physical clusters assigned to them). For the example below, there are 2,197,312 (or 0x218740) clusters that are sparse and only 8304 (or 0x2070) clusters have data in them.



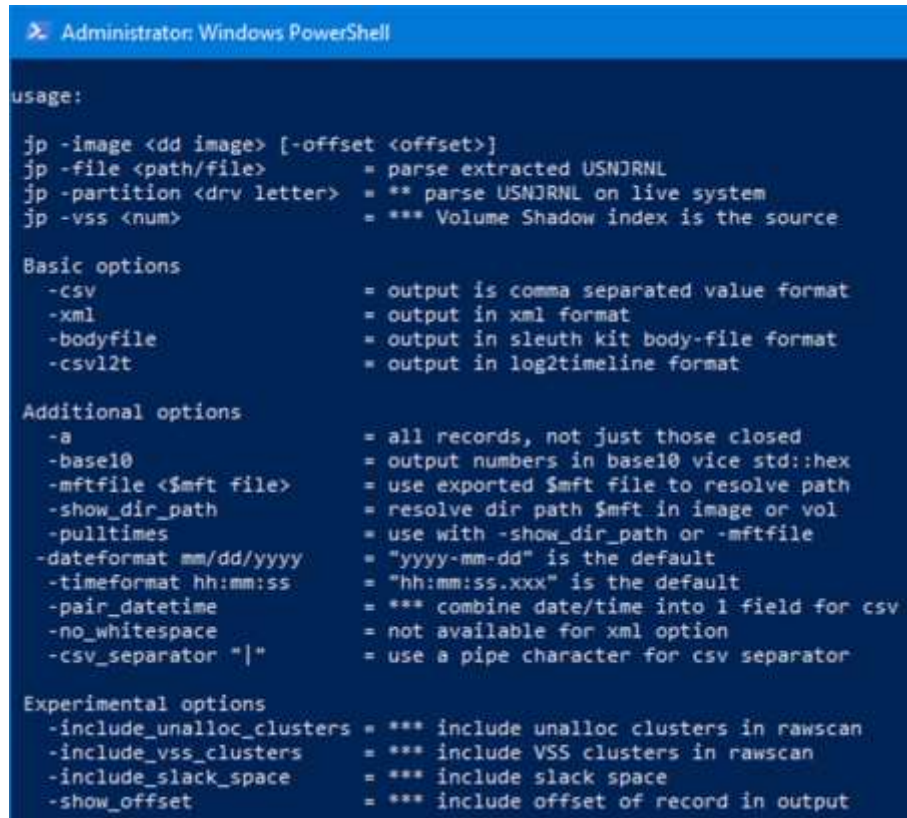
The data within the change journal is a series of packed entries. Each entry is called a *USN_RECORD* structure which is defined in the Microsoft Software Development Kit (SDK). This structure is shown below as documented by the SDK:

```
typedef struct {
    DWORD      RecordLength;
    WORD       MajorVersion;
    WORD       MinorVersion;
    DWORDLONG  FileReferenceNumber;
    DWORDLONG  ParentFileReferenceNumber;
    USN        Usn;
    LARGE_INTEGER TimeStamp;
    DWORD      Reason;
    DWORD      SourceInfo;
    DWORD      SecurityId;
    DWORD      FileAttributes;
    WORD       FileNameLength;
    WORD       FileNameOffset;
    WCHAR      FileName[1];
} USN_RECORD, *PUSN_RECORD;
```

While *jp* extracts all the data from each record, it outputs the following fields: (a) *FileReferenceNumber* (also referred to as MFT entry or inode for the file or directory), (b) *ParentFileReferenceNumber* (which is the parent inode), (c) *Usn* number (which translates to the offset within the data stream), (d) *TimeStamp* (UTC date/time when the record was entered), (e) *Reason* (what change occurred to the file or directory that caused a journal entry), (f) *FileAttributes*, and (g) *Filename*. *jp* can also pull data from the \$MFT and other sources to provide a clearer picture of where the entry came from in the filesystem and other timestamps associated with the entry. This will be discussed in later sections.

3 How to Use *jp*

For live extraction and analysis, the *jp* tool requires one to run with administrator privileges; without doing so will restrict one to only looking at previously extracted change log journals. One can display the menu options by typing in the executable name without parameters. A screen shot of the menu is shown below.



```
Administrator: Windows PowerShell

usage:

jp -image <dd image> [-offset <offset>]
jp -file <path/file>                = parse extracted USNJRNL
jp -partition <drv letter>          = ** parse USNJRNL on live system
jp -vss <num>                      = *** Volume Shadow index is the source

Basic options
-csv                               = output is comma separated value format
-xml                               = output in xml format
-bodyfile                         = output in sleuth kit body-file format
-csv12t                           = output in log2timeline format

Additional options
-a                                 = all records, not just those closed
-base10                           = output numbers in base10 vice std::hex
-mftfile <$mft file>              = use exported $mft file to resolve path
-show_dir_path                    = resolve dir path $mft in image or vol
-pulltimes                        = use with -show_dir_path or -mftfile
-dateformat mm/dd/yyyy            = "yyyy-mm-dd" is the default
-timeformat hh:mm:ss              = "hh:mm:ss.xxx" is the default
-pair_datetime                    = *** combine date/time into 1 field for csv
-no_whitespace                    = not available for xml option
-csv_separator "|"                = use a pipe character for csv separator

Experimental options
-include_unalloc_clusters         = *** include unalloc clusters in rawscan
-include_vss_clusters             = *** include VSS clusters in rawscan
-include_slack_space              = *** include slack space
-show_offset                      = *** include offset of record in output
```

From the menu above, there are three primary data source options: (a) input from an extracted journal file, (b) a *dd* image of a volume or disk, and (c) a mounted partition of a live Windows machine. *jp* can handle each equally well. The latter two, however, actually can yield more data. Specifically, by allowing *jp* to look across an entire volume, it can cross-reference the MACB timestamp data from INDX slack space when analyzing deleted entries. Secondly, *jp* can analyze older journal entries that have been deleted. There are 3 sets of options that can find these entries (see *-include_unalloc_clusters*, *-include_vss_clusters* and *-include_slack_space*).

All the data source options allow one to reconstruct the parent path of the journal entry, using the *-show_dir_path* switch. This is useful for identifying where the target file or directory was from. As a separate option, *jp* allows one to explicitly point to a \$MFT exported file to use for path reconstruction (ref: *-mftfile <\$MFT file>*). If *jp* points to a volume (via the *-partition* option) or an image (via the *-image* option), then the tool will use the associated embedded \$MFT file to extract the necessary metadata, however if an external \$MFT is provided via the *-mftfile* switch, then the external \$MFT file will take precedence.

The \$MFT file is also useful for extracting standard information MACB times that are related to the journal entry. This can be done via the **-pulltimes** option. Normally, this option just pulls the MACB times from the appropriate \$MFT entry. However, if there is no \$MFT entry, **jp** will perform additional analysis and start looking at the slack space of the parent directory's INDX records to see if it can find a matching entry. If it finds a matching entry it will extract the MACB timestamps from the slack and report it in the output as *deleted/wisp*, to indicate it used the TZWorks **wisp** engine to extract the results.

The term **wisp** refers to another TZWorks tool that targets INDX slack entries. Some functionality of the **wisp** tool was added to **jp** and hence the term used in the output. The **wisp** engine portion of **jp**, however, only works if it has access to the entire volume when doing its analysis.

3.1 Processing 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 Windows Vista up to Win10. It does not apply to Windows XP.

There are 2 options available to pointing to the change log journal on a Volume Shadow. The first option follows the format of the other tools and uses the built in shortcut syntax to access 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 hive. For example, to access a System hive located on *HarddiskVolumeShadowCopy1*, the following syntax can be used:

```
jp -file %vss%1\$\Extend\$\UsnJrnl:$J > results.txt
```

The second option uses the option is much easier and uses the **-vss <index of Volume Shadow>** syntax. Below yields the same result as the first one above.

```
jp -vss 1 > results.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, 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 previous options.

3.2 Including old entries from Unallocated Clusters

Starting with version 1.18, if one wants to scan all the unallocated clusters, one can issue the option: *-include_unalloc_clusters*, in combination with one of these options: *-image*, *-partition* or *-vss*.

Using the *-include_unalloc_clusters* option, *jp* will first scan the normal *\$UsnJrnl:\$J* location and then proceed to scan all the unallocated clusters for old change log entries. The output will be annotated with another column titled "unalloc" to specify which change log entry was found in the unallocated cluster section and which was not.

Originally we thought this would be more difficult than it was, since the change log journal doesn't have a magic signature per se. So using some customized fuzzy logic, we added the option to scan unallocated clusters and pull out old change log journal entries. After some quick tests, surprisingly, there are a number of entries that are available and can be extracted successfully. We tried to tune the scanning to minimize false-positives at the expense of missing some valid entries. While this adds a useful option, it should be considered experimental.

For those analysts that would like to verify these entries, one can use the *-show_offset* option. This will instruct *jp* to show the volume offset of where the entry came from. Then one can manually review and validate those entries with a hex viewer.

3.3 Including old entries from Volume Shadow Clusters and Slack space

Starting with version 1.24 and 1.25, one the options *-include_vss_clusters* and *-include_slack_space* were added respectively. Both options are carving options, in that they look for USNJRNL signatures as the precursor to parsing. They can be used together and in conjunction with the *-include_unalloc_clusters* to maximize the number of USNJRNLs in a specified volume.

3.4 Available Output Options

There are four output format options available, ranging from: (a) the default CSV output, (b) XML format, (c) Log2Timeline format, and (d) Bodyfile format defined by the Sleuth Kit. The default option, at this time, yields the most data per record and provides the more verbose data. Specifically, the additional timestamp data is only displayed with the default option. The Log2Timeline is geared for timeline analysis.

When analyzing the change log journal data, one can view only those records that have been closed. Closed, here, means the journaling entry was completed and recorded successfully. This is the default output. If one wants to see 'all' the records, one can use the *-a* switch. This will yield a lot of

redundant data since it will show the journal action before the 'closed' action as well as the 'closed' action.

3.4.1 CSV Output

As stated earlier, the CSV default option yields the most data. These fields are best described by a picture of the output. For this example, the default option CSV option was used and by looking at the column headers, one gets a good idea of the fields that can be extracted from each journal entry.

usn	date	time-UTC	MFT entry	seqnum	parent MFT	seqnum	usn#	attributes	filename	type change
10/22/2013	16:29:18.835	0x0000be91	0x0004	0x0000714c	0x0006	0x4be8148c0	archive; nc	SDelete[1].zip	data_append	
10/22/2013	16:29:18.841	0x0000be91	0x0004	0x0000714c	0x0006	0x4be814918	archive; nc	SDelete[1].zip	file_deleted	
10/22/2013	16:29:18.880	0x0000be92	0x0003	0x00000fb0	0x0002	0x4be814b30	archive	SDelete.zip	ads_data_ap	
10/22/2013	16:29:18.882	0x0000be92	0x0003	0x00000fb0	0x0002	0x4be814be0	archive	SDelete.zip	access_change	
10/22/2013	16:29:46.640	0x0000be94	0x0004	0x00000fb0	0x0002	0x4be817050	dir	SDelete	file_created	
10/22/2013	16:29:46.781	0x0000be95	0x0003	0x0000be94	0x0004	0x4be817150	archive	sdelete.exe	data_append	
10/22/2013	16:29:46.781	0x0000be95	0x0003	0x0000be94	0x0004	0x4be817200	archive	sdelete.exe	basic_info	
10/22/2013	16:29:46.900	0x0000be95	0x0003	0x0000be94	0x0004	0x4be817308	archive	sdelete.exe	ads_data_app	
10/22/2013	16:29:46.912	0x0000be96	0x0003	0x0000be94	0x0004	0x4be817400	archive	Eula.txt		

Basic output

When using path reconstruction and adding times to the output, the data can make a long record. For this example, we display the same entries as the above example, but broke it up into two screenshots. The first image is the same as the one above, and the second image (below) is the extra data that shows the path reconstruction and timestamps, if they were available.

To make the example more interesting, we targeted a journal record that was deleted and **jp** was able to extract information about the deleted entry from the slack data of the parent directory's INDX record. The **wisp** recovered entry is highlighted in the figure.

filename	MFT statu	mdate	time-UTC	act	Trcdate	time-UT	Path
SDelete[1].zip	del/entry						[root]\Users\Donald\AppData\Local\Pa
SDelete[1].zip	del/entry						[root]\Users\Donald\AppData\Local\Pack
SDelete.zip	del/wisp	10/22/2013	16:29:18.		10/22/2013	16:29:18	[root]\Users\Donald\Downloads\
SDelete.zip	del/wisp	10/22/2013	16:29:18.		10/22/2013	16:29:18	[root]\Users\Donald\Downloads\
SDelete	del/entry						[root]\Users\Donald\Downloads\
sdelete.exe	del/parent						parent dir and file no longer ref'd in \$MFT
sdelete.exe	del/parent						parent dir and file no longer ref'd in \$MFT
sdelete.exe	del/parent						parent dir and file no longer ref'd in \$MFT
Eula.txt	del/parent						

During any path reconstruction, where there are deleted entries and old inode numbers have been recycled into new directories and/or files, there exists the issue of handling false positives. Therefore, starting with version 1.09, **jp** looks at both the inode and sequence number to determine if there is a match (for both the parent and target entry). To try to make it clearer to the analyst, the following data is annotated using the nomenclature in the table below to each entry.

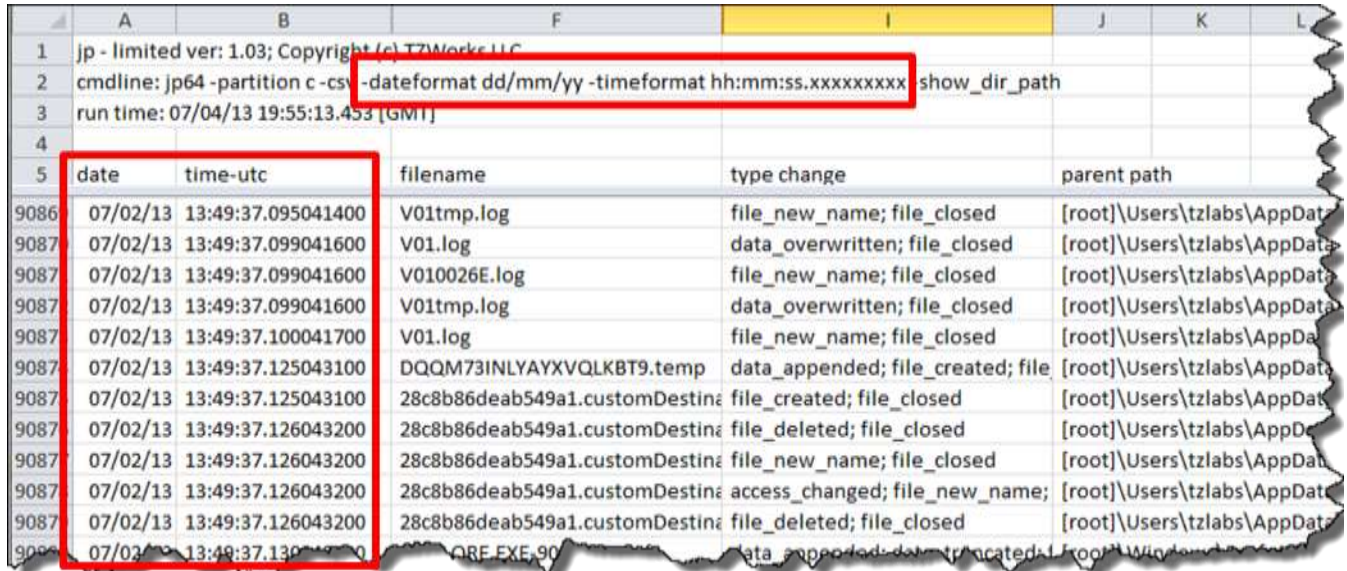
Name	Meaning
valid	Journal entry found in \$MFT
del/entry	Journal entry not present in \$MFT, but parent entry was
del/parent	Journal entry's parent not present in \$MFT
del/wisp	Journal entry not present in \$MFT, but parent entry was and wisp engine was able to extract a matching Journal entry inode/sequence number pair.

3.4.2 Time resolution and Date format

Another feature is the ability to change the date and/or time format to conform to whatever standard that is desired, with a few restrictions. The time format allows one to show as much (or as little) precision that the Windows native FILETIME allows, which is documented to have a resolution of 100 nanoseconds.

Since the FILETIME format allows for 100 nanosecond resolution, one can display this resolution if desired. The **-timeformat** field allows one to throttle this by just adding more *x*'s to the end of the template argument (*hh:mm:ss.xxx*). This becomes more important when you want to determine the order of events when they are closely aligned in time, as is the change journal entries.

Also shown in the example, is the **-dateformat** was used to modify the default *mm/dd/yyyy* format to a *dd/mm/yy* format.



	A	B	F	I	J	K	L
1	jp - limited ver: 1.03; Copyright (c) TZWorks LLC						
2	cmdline: jp64 -partition c -csv -dateformat dd/mm/yy -timeformat hh:mm:ss.xxxxxxxx show_dir_path						
3	run time: 07/04/13 19:55:13.453 [GMT]						
4							
5	date	time-utc	filename	type change	parent path		
9086	07/02/13	13:49:37.095041400	V01tmp.log	file_new_name; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.099041600	V01.log	data_overwritten; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.099041600	V010026E.log	file_new_name; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.099041600	V01tmp.log	data_overwritten; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.100041700	V01.log	file_new_name; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.125043100	DQQM73INLYAYXVQLKBT9.temp	data_appended; file_created; file	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.125043100	28c8b86deab549a1.customDestina	file_created; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.126043200	28c8b86deab549a1.customDestina	file_deleted; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.126043200	28c8b86deab549a1.customDestina	file_new_name; file_closed	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.126043200	28c8b86deab549a1.customDestina	access_changed; file_new_name;	[root]\Users\tzlabs\AppData		
9087	07/02/13	13:49:37.126043200	28c8b86deab549a1.customDestina	file_deleted; file_closed	[root]\Users\tzlabs\AppData		
9088	07/02/13	13:49:37.130000000	QRE EXE, 90	data_appended; data_truncated;	[root]\Windows		

4 Change Journal Reason Code

When looking at the *type of change* (or *reason code*) that caused the journal entry, there are 21 possibilities. Each possibility is enumerated below along with how it is mapped to *jp*.

The source of the reason codes came from the Microsoft Software Development Kit (SDK).

Reason code	Reason label	<i>jp</i> output	MACB mapping	Description
0x00008000	USN_REASON_BASIC_INFO_CHANGE	basic_info_changed	MACB	A user has either changed one or more file or directory attributes or one or more time stamps.
0x80000000	USN_REASON_CLOSE	file_closed	-	The file or dir was closed.
0x00020000	USN_REASON_COMPRESSION_CHANGE	compression_changed	MACB	The compression state of the file or directory was changed.
0x00000002	USN_REASON_DATA_EXTEND	data_appended	MACB	The file's unnamed stream had data appended to it.
0x00000001	USN_REASON_DATA_OVERWRITE	data_overwritten	MACB	The unnamed stream in the file or dir had its data overwritten.
0x00000004	USN_REASON_DATA_TRUNCATION	data_truncated	MACB	The file's unnamed stream was truncated.
0x00000400	USN_REASON_EA_CHANGE	extended_attrib_changed	MACB	The user made a change to the file or directory extended attributes..
0x00040000	USN_REASON_ENCRYPTION_CHANGE	encryption_changed	MACB	The file or directory was encrypted or decrypted.
0x00000100	USN_REASON_FILE_CREATE	file_created	MACB	The file or dir was created.
0x00000200	USN_REASON_FILE_DELETE	file_deleted	MACB	The file or dir was deleted.
0x00010000	USN_REASON_HARD_LINK_CHANGE	hardlink_changed	MACB	An NTFS hard link was added to or removed from the file.
0x00004000	USN_REASON_INDEXABLE_CHANGE	context_indexed_changed	MACB	A user changed the FILE_ATTRIBUTE_NOT_CONTENT_INDEXED attribute.
0x00000020	USN_REASON_NAMED_DATA_EXTEND	ads_data_appended	MACB	The named stream in the file or directory had data appended to it.
0x00000010	USN_REASON_NAMED_DATA_OVERWRITE	ads_data_overwritten	MACB	The named stream in the file or directory had its data overwritten.
0x00000040	USN_REASON_NAMED_DATA_TRUNCATION	ads_data_truncated	MACB	The named stream in the file or directory was truncated.
0x00080000	USN_REASON_OBJECT_ID_CHANGE	objid_changed	MACB	The object identifier of the file or directory was changed.
0x00002000	USN_REASON_RENAME_NEW_NAME	file_new_name	MACB	The file or directory was renamed, and the file name in this record is the new name.
0x00001000	USN_REASON_RENAME_OLD_NAME	file_old_name	MACB	The file or directory was renamed, and the file name in this record is the old name
0x00100000	USN_REASON_REPARSE_POINT_CHANGE	reparse_changed	MACB	The file or directory reparse point attribute was added, changed, or deleted.
0x00000800	USN_REASON_SECURITY_CHANGE	access_changed	MACB	A change was made in the access rights to the file or directory.
0x00200000	USN_REASON_STREAM_CHANGE	ads_added_or_deleted	MACB	A named stream has been added, renamed, or removed from the file or directory.

In the above table, the mapping to MACB is defined differently than the normal MACB times used for file times. Since the 'A' label which normally stands for 'access time' does not get used in the journaling flags, it has been redefined to mean 'data appended'. Also the 'M' and 'C', which normally stands for 'modify time' and 'system metadata changed time', respectively, are now defined as follows: 'M' is defined for 'data overwrite' which means the file has completely changed in content, and 'C' is for the rest of the changes to the file (whether it be file data or system metadata). Breaking it out this way adds fidelity in the results when analyzing the MACB for interesting changes.

5 Known Issues

For csv (comma separated values) output, there are restrictions in the characters that are outputted. Since commas are used as a separator, any data that had a comma in its name is changed to a semicolon. For the default (non-csv) output, no changes are made to the data. One can use the **-csv_separator “|”** option to switch the default comma delimiter to a pipe delimiter to solve this issue.

6 Available Options

Option	Description
-file	Extract artifacts from an extracted \$UsnJrnl:\$J file.
-partition	Extract artifacts from a mounted Windows volume. The syntax is -partition <drive letter> .
-image	Extract artifacts from a volume specified by an image and volume offset. The syntax is -image <filename> -offset <volume offset>
-vss	Experimental. Extract change log journal 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.
-xml	Outputs results in XML format.
-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.

-a	Option to display 'all' records, not those just closed. The default option is to only display entries that are closed.
-show_dir_path	Option to resolve the directory path. This option is not available with the -file option.
-mftfile	Use data in exported \$MFT file to resolve the directory path. The syntax is -mftfile <exported \$mft file>
-pulltimes	If either the -show_dir_path or -mftfile options are specified, this option will extract the appropriate MACB times from the \$MFT and report them with the journal entry parsed. As a special case, if the \$MFT is coming from a specified volume (via the -partition option) or via an image (via the -image option), and if the journal entry is not found the in \$MFT file, it will scan the entry's parent directory slack INDX records for the MACB times.
-base10	Ensure all size/address output is displayed in base-10 format vice hexadecimal format. Default is hexadecimal format.
-no_whitespace	Used in conjunction with -csv option to remove any whitespace between the field value and the CSV separator.
-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.
-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.xxxxxxxxxx" , 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
-include_unalloc_clusters	Experimental. Scan unallocated clusters as well. Results may include false positives. If one wants to see the offset of the entry, use the sub-option -show_offset
-include_vss_clusters	Experimental. Scan all Volume Shadow clusters. Results may include false positives. If one wants to see the offset of the entry, use the sub-option -show_offset
-include_slack_space	Experimental. Scan MFT data's slack space for USNJRNL entries. Results may include false positives. If one wants to see the offset of the entry, use the sub-option -show_offset
-utf8_bom	All output is in Unicode UTF-8 format. If desired, one can prefix an UTF-8 <i>byte order mark</i> to the CSV output using this option.

7 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.

7.1 *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.

8 References

1. Microsoft Systems Journal. Keeping an Eye on your NTFS Drives: the Windows 2000 Change Journal Explained, by Jeffrey Cooperstein and Jeffrey Richter, 9/99
2. Various MSDN articles.
3. Briefing on Advanced \$UsnJrnl Forensics, by BlueAngel, ref: <http://forensicinsight.org/wp-content/uploads/2013/07/F-INSIGHT-Advanced-UsnJrnl-Forensics-English.pdf>