

TZWorks® CSV Data eXchange (*csvdx*) Users Guide



Abstract

csvdx is a standalone, command-line tool that takes a file that has delimited data (such as a CSV file) and allows one the option to convert it into HTML, JSON, or SQLite format. **csvdx** understands the unique headers produced in other TZWorks® tools to ensure maximum data retention during the conversion. There are compiled versions for Windows, Linux and macOS.

Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v0.43 of **csvdx**

Updated: Apr 25, 2025

Table of Contents

1	Introduction	2
1.1	Handling CSV Data in its Various Forms	2
1.2	Extra Data in the Output of TZWorks Tools.....	4
1.3	Handling Delimiters in the Raw Field Data	4
1.4	Handling Large CSV Files.....	5
2	How to Use <i>csvdx</i>	6
2.1	Manipulating the CSV Data.....	7
2.2	HTML Output	7
2.3	JSON Output	9
2.4	SQLite Output	9
2.4.1	SQLite Dependencies	10
2.4.2	Schema of the SQLite Database	10
2.4.3	Combining Multiple CSV files into one Database.....	11
2.4.4	Converting the SQLite Output into Meaningful CSV files	13
2.5	Splitting a CSV file into Separate Files	14
3	Available Options	17
3.1	General Options.....	17
3.2	CSV Specific Sub-Options.....	18
3.3	SQLite Specific Sub-Options.....	19
4	Authentication and the License File.....	20
4.1	<i>Limited</i> versus <i>Demo</i> versus <i>Full</i> in the tool's Output Banner	20
5	References	20

TZWorks® CSV Data eXchange (*csvdx*) Users Guide

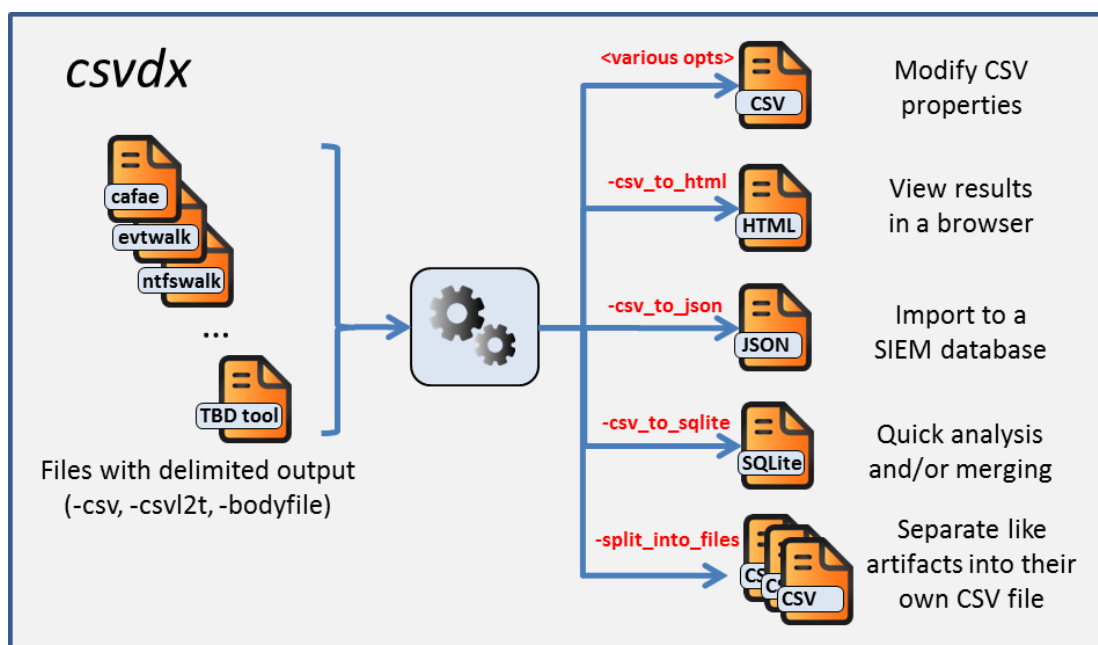
Copyright © TZWorks LLC

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=34

Contact Information: info@tzworks.com

1 Introduction

csvdx is a command line, support tool that converts delimited data (such as CSV data) into other formats. Currently **csvdx** supports conversion to: (a) HTML table data, (b) JSON format, and (c) a SQLite database. These formats are useful if desiring to: (a) displaying the data in other viewers (b) importing the original delimited data to other databases, or (c) just trying to merge similar artifacts together. Pictorially the functionality of **csvdx** is shown below:



1.1 Handling CSV Data in its Various Forms

CSV stands for Comma Separated Values. However, in this document, the term CSV will also be used to refer to other delimiters as well, such as **tab** delimited, **pipe** character delimited, etc. Currently, **csvdx** can handle the following delimited data:

- Comma character**

- b. **Pipe character**
- c. **Tab character**

In addition to the delimiter, one needs to take into account the end of line (EOL) character(s) – called here as the EOL symbol. It is this EOL symbol that distinguishes between one record and another record. Unfortunately, the EOL symbol can vary depending on which operating system the CSV file was generated on. Below are the types of EOL symbols used, based on operating system type.

Operating System	EOL name	EOL Abbreviation	EOL character(s)
Unix & macOS	Line Feed	LF	0x0a
Windows	Carriage Return / Line Feed	CRLF	0x0d 0x0a
Older Mac (OS 9)	Carriage Return	CR	0x0d

For both the field delimiter and EOL symbol, **csvdx** can recognize the type of delimiter and EOL symbol, on the fly, when reading the CSV file.

Finally, the last nuance is the handling the actual text format of the CSV file. Original csvdx was designed to parse UTF-8 formatted CSV files, but later starting with v.0.18 can handle UTF-16 formatted files as well. This assumes, however, that the UTF-16 formatted file is using a Byte Order Mark (BOM) of 0xff, 0xfe at the start of the document. This particular BOM is for little endian UTF-16. **csvdx** will look at the header bytes of the document and if a BOM is present will adjust accordingly to its parsing. Currently **csvdx** cannot handle BOM for UTF-16 big endian or those for UTF-32.

1.2 Extra Data in the Output of TZWorks Tools

The default behavior for tools built by TZWorks is to generate a banner at the top of the file before proceeding with any delimited data. This data contains some additional information that can be useful, if retained, when converting the delimited data to another format. Information such as: (a) the command line options used to parse the original artifact, (b) the timestamp when the parsing was done, (c) the license /organization that conducted the parsing, and (d) which version of the tool was used. **csvdx** reads this banner data and subsequently embeds it to the converted format so it is preserved.

The other, non-standard CSV data that may be present in TZWorks® tools is when processing differing artifact types and storing the results in one CSV file. In these cases, the differing artifacts may have different columns which correspond to the different fields of artifact being processed. Good examples of this are when processing registry data via **cafae** or processing event logs with **evtwalk**. In both cases, the resulting CSV file will have multiple CSV sections. To handle this, **csvdx** looks at the banner data and adjusts the parsing logic based on the tool (which is recorded in the banner) that was used to generate the CSV file. When using the SQLite option to store the artifact data from the CSV file, the banner data will allow **csvdx** to break the data out by artifact within the SQLite database. One can later separate out each individual artifact type using the **-artifact_tables** option, which is discussed later in this guide.

If, on the other hand, there is no banner information, but just delimited data, then **csvdx** will use the first delimited line as the header and treat the rest of the delimited lines as records. This implies, also, that if the delimited data is mixed and disjointed with varying fields, **csvdx** will yield unpredictable results.

1.3 Handling Delimiters in the Raw Field Data

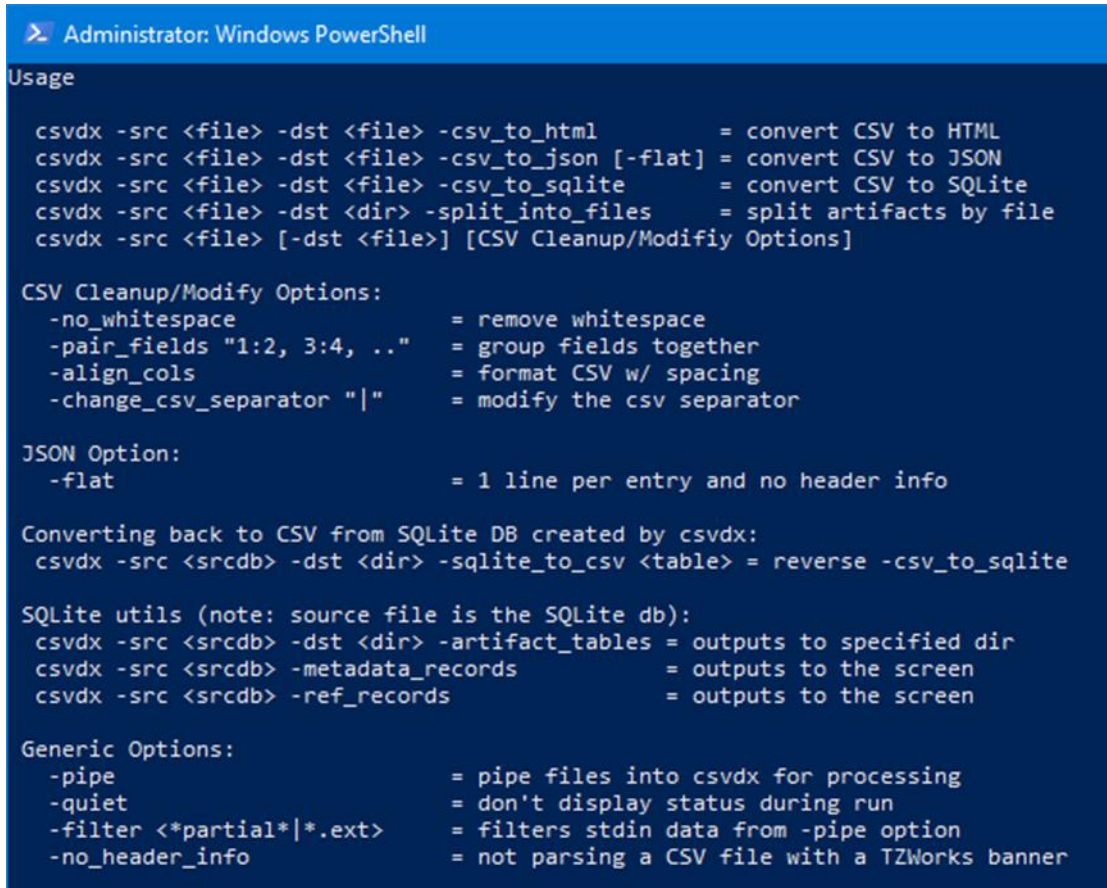
One of the issues with CSV formats is that the delimiter used to separate the fields may actually be in the raw field data itself. The result of this is the record appears to have more fields than the number of column headers and the data is shifted as a result of the extra delimiters. The rules, **csvdx** uses to handle this type of situation, is as follows: (a) the number of delimiters is determined by the number of column headers, (b) if any records following the header record have more delimiters than the header record, the extra fields (from the perspective of the **csvdx**) are combined together into an overflow field. The intent here is to try to preserve all the data, but limit the disruption to the record by adding one overflow field. For those cases where a record has less delimiters than the header record, then **csvdx** treats this as a possible record that had a EOL symbol embedded into one of its fields. To try to correct this, **csvdx** will read the next record and see if it can logically reconstruct the broken record into one. It determines this by adding the number of delimiters between the 2 records, and seeing if, by combining them together, they total the number of delimiter in the header record. If so, then they are combined; if not, then it is outputted as a broken record.

1.4 Handling Large CSV Files

The final concern with creating a tool to convert CSV files into another format is the issue of handling very large CSV files. Processing artifacts from some of the 64 bit operating systems and their ability to handle very large files and large storage devices can yield some hefty output files that would be need to be handled. A good example is looking at event logs from a Win7 box on up. Eventlogs can yield 4 GB of data easily and can be the norm on some systems that care about logging many things. To handle very large files, **csvdx** has been designed to read input files in chunks and processing them on a chunk by chunk basis. This approach allows **csvdx** to handle very large CSV files.

2 How to Use *csvdx*

Below is a screen shot of the command line menu. This shows all the options in summary form.



```
Administrator: Windows PowerShell

Usage

csvdx -src <file> -dst <file> -csv_to_html      = convert CSV to HTML
csvdx -src <file> -dst <file> -csv_to_json [-flat] = convert CSV to JSON
csvdx -src <file> -dst <file> -csv_to_sqlite     = convert CSV to SQLite
csvdx -src <file> -dst <dir> -split_into_files   = split artifacts by file
csvdx -src <file> [-dst <file>] [CSV Cleanup/Modify Options]

CSV Cleanup/Modify Options:
-no_whitespace           = remove whitespace
-pair_fields "1:2, 3:4, .." = group fields together
-align_cols              = format CSV w/ spacing
-change_csv_separator "|" = modify the csv separator

JSON Option:
-flat                    = 1 line per entry and no header info

Converting back to CSV from SQLite DB created by csvdx:
csvdx -src <srcdb> -dst <dir> -sqlite_to_csv <table> = reverse -csv_to_sqlite

SQLite utils (note: source file is the SQLite db):
csvdx -src <srcdb> -dst <dir> -artifact_tables = outputs to specified dir
csvdx -src <srcdb> -metadata_records           = outputs to the screen
csvdx -src <srcdb> -ref_records                = outputs to the screen

Generic Options:
-pipe                = pipe files into csvdx for processing
-quiet               = don't display status during run
-filter <[*partial*|*.ext]> = filters stdin data from -pipe option
-no_header_info      = not parsing a CSV file with a TZWorks banner
```

For all options, a source file must be specified to operate on, via the **-src** switch. The output can either be sent to another file via the **-dst** switch or to the screen (without the **-dst** switch). The main options are **-csv_to_ [json | sqlite | html]**.

If not desiring to change the CSV, to some different format, one can retain the delimited nature, and just modify some of the CSV properties, via the CSV Cleanup/Modify options. These include:

(a) **-no_whitespace**, to remove any spaces around the delimiters, (b) **-pair_fields**, to combine 2 fields in the original CSV data into one field, (c) **-align_cols**, to space out the fields so they are better aligned for readability in a text editor, and (d) **-change_csv_separator**, to change the existing field separator to something else.

If *SQLite* is the desired conversion, there are other options that can be used. Since **csvdx** dynamically creates tables on the fly based on each unique CSV field set, one can pipe in multiple CSV files into one *SQLite* database, via the **-pipe** option. Each CSV file can represent a separate forensic artifact, and the piping operation will retain the uniqueness of the CSV artifact data across files. More information

about the SQLite format and what can be done from then on is discussed in the section under SQLite data.

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.

2.1 Manipulating the CSV Data

If one is given a CSV file that has formatting properties that need to be changed, **csvdx** offers 4 possible options to modify the CSV file via the following switches:

1. **-no_whitespace** to remove all white space (space and tabs) between the field values and CSV separators. This is the same option available in most TZWorks® tools.
2. **-pair_fields "<field1>:<field2>"** to merge two separate fields into one field. Where one may want to do this is when one field is the date field and another field is the time field, and what is desired is one field containing both the date and time. If the date field is at position 1 and the time field is at position 2, the command **-pair_fields "1:2"** will merge the date and time (separated by a space) into one field.
3. **-align_cols** to put spaces in between field data and the delimiters. This purpose here would be to align all the data, so that the CSV output is easy to read with notepad or some other text viewer.
4. **-change_csv_separator "<char>"** to change the existing CSV delimiter to something else. What this option does under the covers is if using either a pipe character or a comma character, it will be able to do the modification while preserving the CSV format, even if the raw data in the fields contain the resulting delimiter. It does this by scanning all the raw field data and takes the following actions: (a) if the resulting delimiter is a pipe character and if a pipe character is found in the data, the pipe character in the raw field data will be changed to a semicolon, and (b) if the resulting delimiter is a comma, and if a comma is found in the data, the comma in the raw field data will be changed to a space.

For all options above, the CSV output is sent to standard output, unless using the **-dst <results file>** option, where it be sent to the specified file. If using the piping option (**-pipe**), to read in multiple CSV files in succession, then the **-dst <results folder>** needs to be used to specify the directory to put all the modified files. For this latter case, the names of the resulting CSV files will be the same names but put in the folder specified by the **-dst** option.

2.2 HTML Output

When converting from CSV data to a HTML table format, use the **-csv_to_html** option and either redirect the output to a file or use the **-dst <result file>** option. Below is an example of a CSV file

generated by **cafae** on the left, and the resulting HTML that is generated on the right. Notice the differing artifacts in the original CSV file are ported over to their respective HTML table, and the banner information is converted as well.

Starting CSV file

```
cafae - full ver: 0.31; Copyright (c) TZworks LLC
License #1d0bd09d075c9c0 is authenticated for business use
run time: 07/30/2015 18:27:56 [UTC]
"cmdline: cafae64 -hive e:\testcase\hives\ntuser\win8.dblake.NTUSER"
```

Artifact **UserInfo**
Registry key: ntuser.dat\Software\Microsoft\Office\Common

```
reg date [UTC],subkey,value name,value data
09/27/2013 03:04:53.337,Common\Smart Tag,migratedBitValues,01 00 00 00
09/23/2013 19:17:52.102,Common\UserInfo,Company,
09/23/2013 19:17:52.102,Common\UserInfo,UserName,Donald Blake
09/23/2013 19:17:52.102,Common\UserInfo,UserInitials,DB
09/23/2013 19:17:52.102,Common,UID,7a b1 c2 12 22 53 c1 4
09/23/2013 19:17:52.102,Common,UserName,Donald
```

Artifact **UserAssist**
Registry key: ntuser.dat\Software\Microsoft\windows\Curre

```
reg date [UTC],modify date [UTC],sess,run,focus cnt,focus
10/23/2013 03:22:30.785,10/23/2013 03:09:22.665,0,1,4,340203
10/23/2013 03:22:30.785,10/23/2013 02:58:52.503,0,5,8,539329
```

Run stats

Resulting HTML file

```
Tool: cafae - ver: 0.31
License: #1d0bd09d075c9c0 - Dave T; TZWorks
Runtime: 07/30/2015 18:27:56 [UTC]
User input: cafae64 -hive e:\testcase\hives\ntuser\win8.dblake.NTUSER
```

UserInfo

reg date [UTC]	subkey	value name	value data
09/27/2013 03:04:53.337	Common/Smart Tag	migratedBitValues	01 00 00 00
09/23/2013 19:17:52.102	Common/UserInfo	Company	
09/23/2013 19:17:52.102	Common/UserInfo	UserName	Donald Blake
09/23/2013 19:17:52.102	Common/UserInfo	UserInitials	DB
09/23/2013 19:17:52.102	Common	UID	7a b1 c2 12 22 53 c1 4 0e e1 15 2b ea 5a 0d Donald

UserAssist

reg date [UTC]	modify date [UTC]	sess	run	focus cnt	focus msec	subkey
10/23/2013 03:22:30.785	10/23/2013 03:09:22.665	0	1	4	340203	{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}/count
10/23/2013 03:22:30.785	10/23/2013 02:58:52.503	0	5	8	539329	{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}/count

csvdx64 -src win8.dblake.ntuser.csv -csv_to_html > out.html

Command to convert to HTML

2.3 JSON Output

When converting from CSV data to JSON format, use the **-csv_to_json** option and either redirect the output to a file or use the **-dst <result file>** option. Below is an example of a CSV file generated by **cafae** on the left, and the resulting JSON file that is generated on the right. Notice the differing artifacts in the original CSV file are ported over to their respective JSON array, and the banner information is converted as well.

The diagram illustrates the process of converting a CSV file to JSON format using the **csvdx64** tool. It consists of three main parts:

- Starting CSV file:** A screenshot of a CSV file generated by **cafae**. It shows header information like "cafae - full ver: 0.31; Copyright (c) TZworks LLC" and data rows for artifacts such as "UserInfo" and "UserAssist".
- Command to convert to JSON:** A terminal window showing the command: `csvdx64 -src win8.dblake.ntuser.csv -csv_to_json > out.txt`.
- Resulting JSON output:** A screenshot of the resulting JSON file. It shows the converted data for "UserInfo" and "UserAssist" artifacts, with each artifact represented as a JSON array of objects containing registry key details.

Red arrows indicate the flow from the CSV file to the command and then to the JSON output.

2.4 SQLite Output

When converting from CSV data to SQLite format, use the **-csv_to_sqlite** option and specify the SQLite database to be created via the **-dst <SQLite db>** option. Below is a tree-view break out of the database that was generated from parsing data from a CSV file from **cafae**. For each artifact that is found in the CSV data, a unique table is dynamically generated for that specific artifact. There are a few functions that should be of note: (a) **csvdx** has the ability to detect similar artifacts and insert into an existing artifact table if already generated, (b) on subsequent runs of **csvdx**, if using an existing SQLite

database that was generated originally from **csvdx**, the artifacts will be merged into the appropriate like-artifact tables, and (c) on subsequent runs of **csvdx**, one can use completely different CSV tool outputs (eg. one from **cafae**, one from **evtwalk**, etc) and the artifact tables will be preserved.

The screenshot shows a SQLite database interface. On the left, a tree view lists tables: 'ref', 'sqlite_autoindex_ref_1', 'metadata', 'cafae.UserInfo-184b89c5dc5d', 'All Rows', '001', '002', '003', '004', '005', '006', 'cafae.UserAssist-1e77757065d', 'cafae.RecentDocs-ad0382679e', 'cafae.Open_Save_MRU-db802', 'cafae.LastVisitedPidIMRU-a68', and 'cafae.StreamMRU-eab03fc5f6'. On the right, a table preview is shown with columns: '[2-subkey]', '[3-value_na]', and '[4-value]'. The preview data includes rows for 'Common\Smart Tag', 'Common\UserInfo', 'Common\UserInfo', 'Common\UserInfo', 'Common', and 'Common'. Annotations with arrows point to specific parts: 'Support tables' points to the 'ref' and 'sqlite_autoindex_ref_1' tables; 'Approach for dynamically creating table schema uses technique to minimize size and guarantee uniqueness' points to the 'All Rows' table; 'Differing artifacts are in their own table' points to the 'cafae.UserInfo-184b89c5dc5d' table.

While this is nice for those users that are familiar with reviewing a database schema and based on that schema, issuing some SQL statement to extract those components that are of interest, it may not be useful to those that don't have that background. For those users, we created an option to pull all the merged artifact table data into separate CSV files. See the section on: *Converting SQLite output into CSV files* for details and an example.

2.4.1 SQLite Dependencies

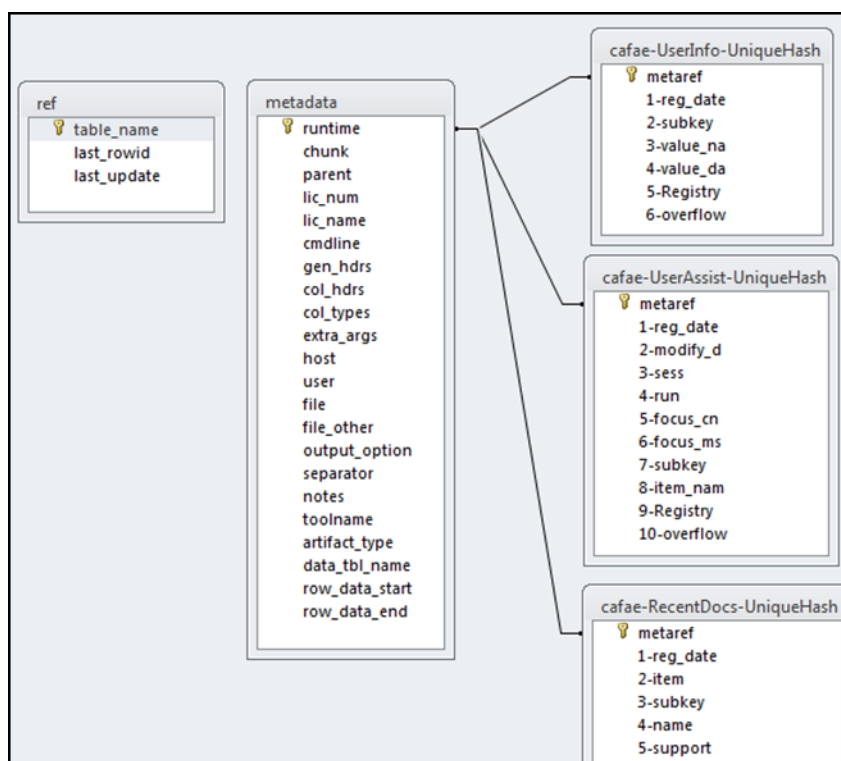
For the SQLite option to work, **csvdx** makes the of the SQLite library. If one is unfamiliar with SQLite, the official SQLite website is <http://www.sqlite.org/>. It has documentation and details on everything one would ever want to know. Starting with version 0.23, the SQLite library is statically linked into **csvdx** binary. What this means is the tool is standalone and does not require any external SQLite shared libraries.

2.4.2 Schema of the SQLite Database

First off, **csvdx** uses 2 baseline tables for keeping track of everything within the database: (1) ref table and the (2) metadata table. The first is used to store off all the tables we are trying to track, including their name, last row-ID and last update time. The second table is used to hold all the metadata for each artifact group that is parsed, including (but not limited to), any banner information that was parsed from the CSV file, column headers associated with the CSV file, the associated artifact table name and the row-ID of the starting record in the artifact table and number of records inserted. The rest of the tables that are generated are strictly based on the number of types of artifacts that are present in the parsed CSV file. The artifact table names are taken from the artifact type specified in the CSV file along with a

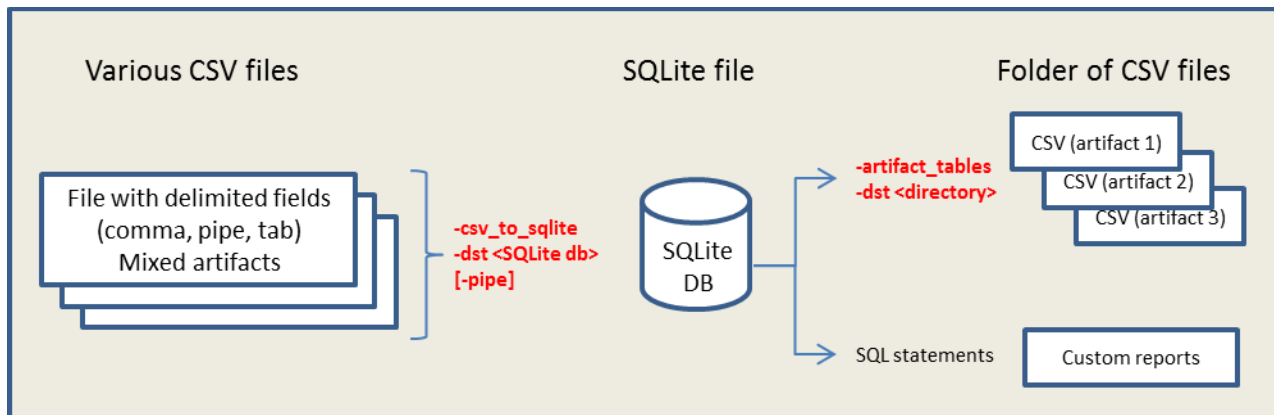
hash to preserve uniqueness in the table schema. Furthermore, the schema used for each artifact table is dependent on the unique field headers used in the CSV file. **csvdx** modifies the column names that were in the CSV file when creating the schema to ensure uniqueness. **csvdx** also truncates the column names so they don't exceed a maximum number of characters and removes any whitespace from the name to ensure they are manageable from a schema standpoint.

Below is an example schema of the SQLite database when running **cafae** on a user hive. The schema only shows three artifact tables as an example. The table names are composed of the tool's name, artifact type, and a unique hash. The word "UniqueHash", in the diagram for artifact tables, is a hash value that is generated for each artifact that takes into account the number and names of the unique fields.



2.4.3 Combining Multiple CSV files into one Database

One can combine multiple CSV files into one database. The way **csvdx** handles this is it starts merging similar artifact data into the appropriate table. This is done a couple of reasons. (a) The first is for efficiency purposes, since the data can still be broken out separately using the metadata table, since it contains a complete mapping of which CSV file data went where. (b) The second reason is this approach also allows for merging of similar artifact data by using the table as a repository for each unique artifact. Unique here is defined to be based on the fields that were archived as well as the banner information specified to identify the artifact type. Below is a diagram showing the overall concept.



For this example we are going to take multiple event log CSV files that were generated from **evtwalk**. When used in the default CSV mode, the output fluctuates as a function of event ID, since each event ID typically has unique fields. Combining multiple CSV files just compounds the problem. Fortunately, **csvdx** is aware of the TZWorks CSV formatting options, and thus, it can successfully interpret all the unique CSV sections and merge them into the appropriate event IDs as represented as separate tables in the resulting SQLite database. There are 2 options available to do this type of merge.

Option 1 is to merge multiple CSV files by adding each one separately to the database by running **csvdx** as a separate session each time. This is useful if the files are not co-located in one subdirectory. For this option, just start with a fresh database name and for subsequent runs continue to use the same database name so all the CSV files are merged into one database. Syntax for each run would be:

```
csvdx -src csvfile1 -dst various.evt.db -csv_to_sqlite
csvdx -src csvfile2 -dst various.evt.db -csv_to_sqlite
...
csvdx -src csvfile2 -dst various.evt.db -csv_to_sqlite
```

Option 2 is to merge multiple CSV files as one **csvdx** session. This assumes all the desired CSV files are in a folder. If this is the case, then all target files can be easily enumerated and piped into **csvdx**.

```
dir <folder with CSV files>/*.csv /b /s | csvdx -pipe -dst various.evt.db -csv_to_sqlite
```

Whatever option is chosen, the resulting SQLite database (various.evt.db, for the example above) will contain a separate table per event ID with the appropriate records populated. To view the tables present one can perform a SQL *select* on the **ref** table, which contains all the table names. The simpler approach is to use the **-ref_records** option. This will not only show the tables present but the number of records per table. As was discussed previously, the table name is decorated with the tool's name, artifact type and a hash of the field names to ensure accurate matching when merging artifacts. See output below from the example performed previously:


```
csvdx64 -src various.evt.db -ref_records > out.txt

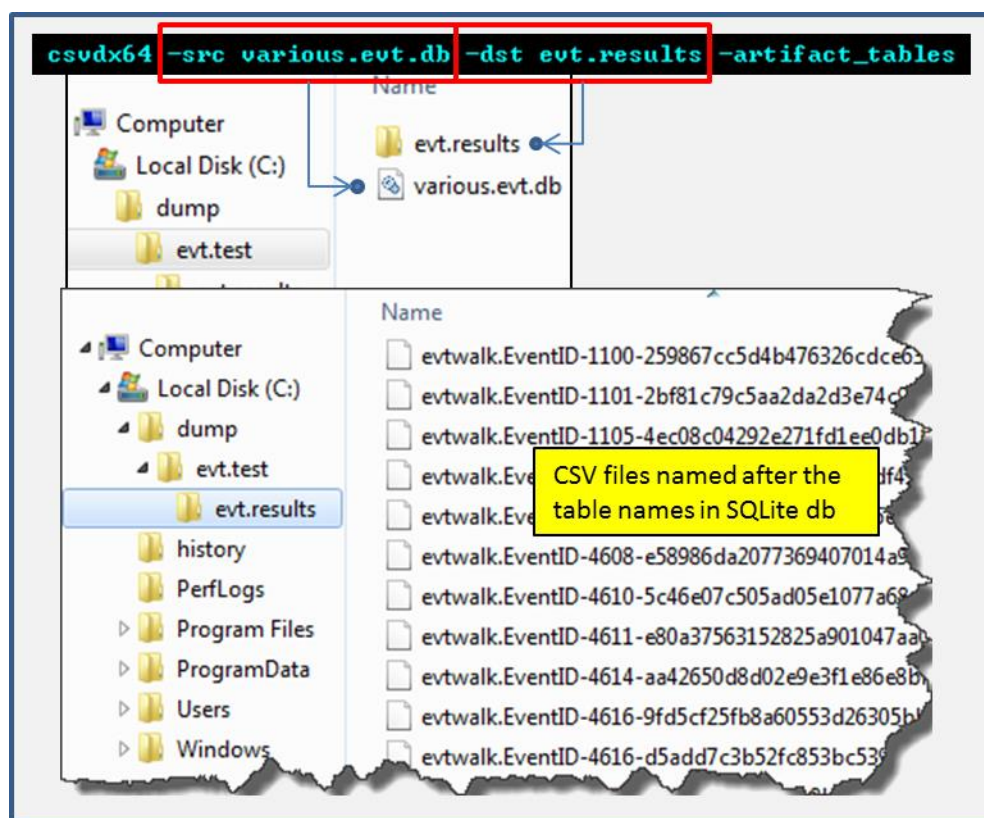
13 : evtwalk.EventID-1100-259867cc5d4b476326cdce
6 : evtwalk.EventID-1101-2bf81c79c5aa2da2d3e74
1 : evtwalk.EventID-1105-42ba57ef35caeca32acdd
2 : evtwalk.EventID-1105-4ec08c04292e271fd1ee
1 : evtwalk.EventID-1108-f5d10a0de34040034832
19 : evtwalk.EventID-4608-e58986da207736940701
3 : evtwalk.EventID-4610-5c46e07c505ad05e1077a
146 : evtwalk.EventID-4611-e80a37563152825a9010
3 : evtwalk.EventID-4614-aa42650d8d02e9e3f1e8
19 : evtwalk.EventID-4616-9fd5cf25fb8a60553d263
5 : evtwalk.EventID-4616-d5add7c3b52fc853bc539
27 : evtwalk.EventID-4617-21e0cc5e3807f150eb1
144 : evtwalk.EventID-4618-21e0cc5e3807f150eb1
34975 : evtwalk.EventID-4619-21e0cc5e3807f150eb1
95 : evtwalk.EventID-4623-21e0cc5e3807f150eb1
34578 : evtwalk.EventID-4634-82e69c436735eaa01ff5
25 : evtwalk.EventID-4647-6c2824047213be5680e45
5184 : evtwalk.EventID-4648-cbbc5fb9fe1f37454a192
209 : evtwalk.EventID-4648-dd01d539bb04f2b9138cbf
1046 : evtwalk.EventID-4656-c9b10938ea6b7db12dbd3
41 : evtwalk.EventID-4658-f0a36a2cbf61456b4406
106 : evtwalk.EventID-4662-74cb148d6bd04dc3aaa63
18617 : evtwalk.EventID-4663-ad532354b1bc8563fcb4
163 : evtwalk.EventID-4670-63122f61700b82222
20523 : evtwalk.EventID-4670-63122f61700b82222
```

Table Names with number of records per table

Note: The merging will have some issues if it senses that the one artifact has some properties that are different than other similar artifacts. For example, looking at the above output, notice EventID-1105, where there are 2 separate tables that are created. The first has 1 item and the second has 2 items. The names are the same, but the hashes are different. This again happens at EventID-4616 and EventID-4648. What this means is one set of records (represented by its own table) has more (or less) fields present in the original CSV output then the other set. More specifically, if one generates a custom CSV file that has some fields present and others that are not, and does not use that template consistently across runs, then the merge operation will detect the differences and store each unique signature set as a separate table.

2.4.4 Converting the SQLite Output into Meaningful CSV files

Hinted in the diagram from the preceding section, if one wanted to extract all the artifact data stored in the SQLite database back into a CSV type output, there is an option called **-artifact tables** to do just that. What this option does internally is: (a) reads the SQLite database specified, (b) extracts all the artifact data while merging the banner specific data pulled during the initial CSV parse with the artifact data, and (c) dumps the final output into a separate CSV files at the directory specified. Therefore, if you had 10 artifact tables to start with, you will end up with 10 unique CSV files with the data from those artifact tables. On the surface, why go through this process of converting some CSV files into SQLite and back out to CSV files? Because, when storing the data into the SQLite database, similar artifacts get grouped together, where they were separated before, so the end result will be all the artifacts have now been outputted as separate CSV artifact files. For the TZWorks® tools this is useful when considering **cafae** and **evtwalk** CSV output data, since the CSV data is mixed in the CSV output. Below is an example of doing this on the SQLite database that was created in the previous section.



After the command is issued, the specified folder (evt.results) will be populated with the CSV files. Each CSV file will use the table name if came from to help provide traceability from which table it came from.

2.5 Splitting a CSV file into Separate Files

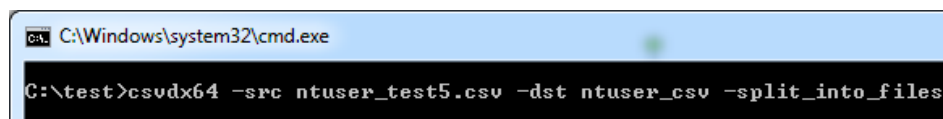
Similar to the functionality just discussed in section on “*Converting the SQLite Output into Meaningful CSV files*,” the goal here is to accomplish similar results without creating a SQLite database in the process. This option, unlike the other ones in **csvdx**, is geared for CSV files containing multiple types of artifacts that are generated by *TZWorks* tools. Specifically, this option will rely on certain header data generated by *TZWorks* tools that get embedded into the CSV reports. However, one can use this option’s merge ability to take similar artifact type CSV reports and merge them all into a single report

As background, some of the *TZWorks* tools render artifacts in a combined CSV report; good examples of tools that do this include **cafae** and **eventwalk**. The reason for this, is creating a combined reports easily allows for the tool(s) to operate across many raw artifact files in a batch processing mode. This is primarily for speed, but also to minimize the footprint of any new files generated on the target box (since our tools are primarily designed for live collection/processing). On the down side, the problem with processing raw artifact files like a registry hive or an event log is they contain many types of differing artifacts. This in turn requires the tool generating the report to create specialized headers per artifact type so it can be understood later during analysis.

Therefore, the new option **-split_into_files** is designed for these report types that have a mixed bag of artifact types interspersed within a CSV file. It will look at each unique artifact type and separate it into a CSV file. Some of the characteristics of this option include:

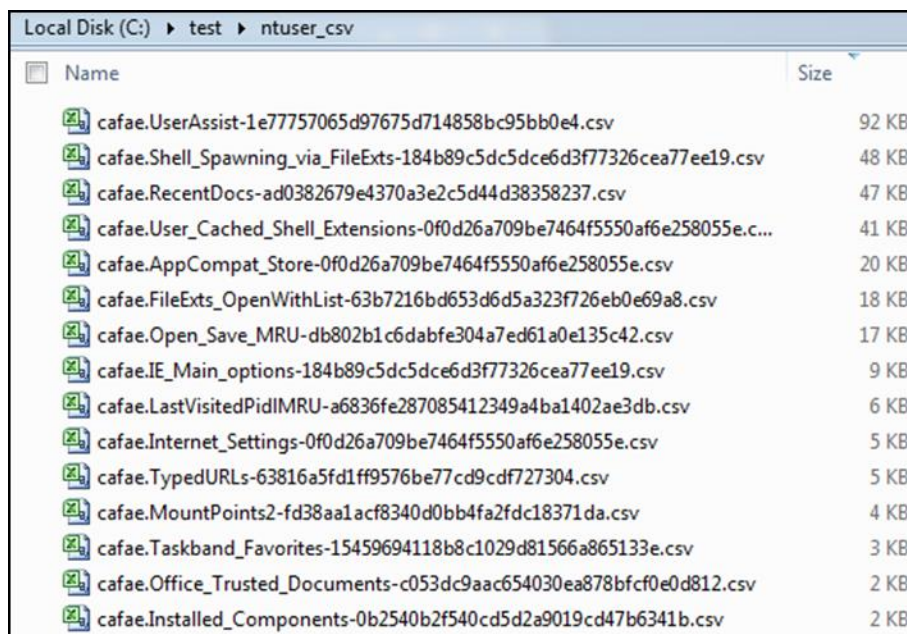
- The similar artifact types can be interspersed throughout the CSV report and should be detected by **csvdx**, so that each unique artifact type will be merged into the appropriate CSV report.
- If the names of the resulting CSV files that are generated are not changed, one can continuously merge additional 'like' artifacts to these files, by just repeating the **csvdx** operation on a new mixed artifact CSV report and specifying the same destination directory as before.
- Due to the merge option above, during the splitting and grouping process, the **TZWorks** banner information will be stripped off and any useful information it contained will be appended to each CSV record. This allows one to continually grow the resulting CSV reports with new data as it comes in. While the additional fields added to each record makes from a larger CSV file, it allows for easier movement later of the resulting CSV data into another database.

Below is an example of the syntax. To specify a CSV to operate on, use **-src <csv file>**. Then one needs to specify a folder to put the resulting CSV files that are generated, via the **-dst <results folder>**. The last parameter is the option to tell csvdx what you want to do, which is **-split_into_files**.



```
C:\Windows\system32\cmd.exe
C:\test>csvdx64 -src ntuser_test5.csv -dst ntuser_csv -split_into_files
```

The destination folder will be created if it is not already present, and the files will be populated as shown below. The filenames are annotated with the tool name, information about the artifact type, and a hash value to indicate the template type the data is using. In this way, the same filename can be referenced for a different **csvdx** session to merge new data into the existing reports (if desired).



As another example, let's say a tool produces a fixed set of field headers for all CSV files, such as the newer *TZWorks* tool **tela**, then the **-split_into_files** option still can provide utility by using looking at a few fields in the data to determine how to group it. For the **tela** use-case, **csvdx** will key off a few fields to gather metadata, such as the "session name" as well as some others. For now, if we focus on just the 'session name' of the log entry, then **csvdx** will use that data to determine which file the data belongs. A raw CSV report from **tela** will look something like the image below. For each file piped into **tela**, it segregates it with a line and then some metadata about the file it parsed. Then it outputs the CSV data for that file. This process repeats until all the files have been processed.

cmdline: <filelist> ... test\tela64.exe -pipe -dateformat yyyy/mm/dd -timeformat hh:mm:ss.xxxxxxxxxx				

file: e:\testcase\etl\win10.sift\LwtNetLog.etl				
orig source path: C:\Windows\System32\LogFiles\WMI\LwtNetLog.etl				
created [utc or #ticks]	provider name	session name	event id	pro
2017/02/24 22:31:06.488888300	Microsoft-Windows-TCPIP	LwtNetLog	0x000529	0x0
2017/02/24 22:31:06.488899300	Microsoft-Windows-TCPIP	LwtNetLog	0x000467	0x
2017/02/24 22:31:06.488902300	Microsoft-Windows-TCPIP	LwtNetLog	0x00045a	0x0
...				






file: e:\testcase\etl\win10.sift\Microsoft-Windows-Kernel-Disk%4Analytic.etl				
orig source path: C:\Windows\System32\Winevt\Logs\Microsoft-Windows-Kernel-Disk%4Analytic.etl				
created [utc or #ticks]	provider name	session name	event id	pro
2017/08/07 00:11:19.158154000	Microsoft-Windows-Kernel-Disk	EventLog-Microsoft-Windows-Kernel-Disk-Analy	0x00000e	0x0
2017/08/07 00:11:19.359461100	Microsoft-Windows-Kernel-Disk	EventLog-Microsoft-Windows-Kernel-Disk-Analy	0x00000e	0x0
2017/08/07 00:11:21.412770200	Microsoft-Windows-Kernel-Disk	EventLog-Microsoft-Windows-Kernel-Disk-Analy	0x00000e	0x00
...				

file: e:\testcase\etl\win10.sift\Microsoft-Windows-USB-UCX-Analytic.etl				
orig source path: C:\Windows\System32\Winevt\Logs\Microsoft-Windows-USB-UCX-Analytic.etl				
created [utc or #ticks]	provider name	session name	event id	pro
2017/08/07 00:07:55.384443900	Microsoft-Windows-USB-UCX	EventLog-Microsoft-Windows-USB-UCX-Analytic	0x000001	0x00
2017/08/07 00:07:55.384446500	Microsoft-Windows-USB-UCX	EventLog-Microsoft-Windows-USB-UCX-Analytic	0x000003	0x00
2017/08/07 00:07:55.384447300	Microsoft-Windows-USB-UCX	EventLog-Microsoft-Windows-USB-UCX-Analytic	0x000045	0x0
...				

file: e:\testcase\etl\win10.sift\Microsoft-Windows-USB-USBHUB3-Analytic.etl				
orig source path: C:\Windows\System32\Winevt\Logs\Microsoft-Windows-USB-USBHUB3-Analytic.etl				
created [utc or #ticks]	provider name	session name	event id	pro
2017/08/07 00:08:17.250760700	Microsoft-Windows-USB-USBHUB3	EventLog-Microsoft-Windows-USB-USBHUB3-Ana	0x000006	0x00
2017/08/07 00:08:17.250764500	Microsoft-Windows-USB-USBHUB3	EventLog-Microsoft-Windows-USB-USBHUB3-Ana	0x000004	0x00
2017/08/07 00:08:17.250764900	Microsoft-Windows-USB-USBHUB3	EventLog-Microsoft-Windows-USB-USBHUB3-Ana	0x000007	0x00
...				

file: e:\testcase\etl\win10.sift\Microsoft-Windows-WMI-Activity%4Trace.etl				
orig source path: C:\Windows\System32\Winevt\Logs\Microsoft-Windows-WMI-Activity%4Trace.etl				
created [utc or #ticks]	provider name	session name	event id	pr
2017/08/07 00:07:45.547846200	Microsoft-Windows-WMI-Activity	EventLog-Microsoft-Windows-WMI-Activity-Trace	0x00000c	0x00
2017/08/07 00:07:45.555793700	Microsoft-Windows-WMI-Activity	EventLog-Microsoft-Windows-WMI-Activity-Trace	0x000032	0x00
2017/08/07 00:07:45.555798600	Microsoft-Windows-WMI-Activity	EventLog-Microsoft-Windows-WMI-Activity-Trace	0x00000c	0x00

So when **csvdx** encounters the above file, it will look at the 'session name' and use that to value to help merge like data. **csvdx** will also record each file parsed by **tela** and annotate that filename in the an extra field on the merged CSV output. That way, the traceability back to the original artifact file is retained.

	tela.session_name-EventLog-Microsoft-Windows-Kernel-Disk-Analytic-8b25761214b26fb89463d04f51ba8e04.csv
	tela.session_name-EventLog-Microsoft-Windows-USB-UCX-Analytic-8b25761214b26fb89463d04f51ba8e04.csv
	tela.session_name-EventLog-Microsoft-Windows-USB-USBHUB3-Analytic-8b25761214b26fb89463d04f51ba8e04.csv
	tela.session_name-EventLog-Microsoft-Windows-WMI-Activity-Trace-8b25761214b26fb89463d04f51ba8e04.csv
	tela.session_name-LwtNetLog-8b25761214b26fb89463d04f51ba8e04.csv

3 Available Options

3.1 General Options

Option	Description
-src	Specifies the source CSV file to operate on. This file will not be modified. Format is -src <source file>
-dst	Specifies the destination file to send the results to. Format is -dst <results file> . Without specifying this option, the results will be outputted to the screen (or can be redirected). In some cases, this option will specify the destination <i>folder (not file)</i> to send the output to.
-csv_to_html	Instructs conversion to be from a CSV file to HTML table format.
-csv_to_json	Instructs conversion to be from a CSV file to JSON format. There is a sub-option -esc_path to allow for backslashes to be preserved with the escape character. Default behavior converts backslashes to forward slashes.
-csv_to_sqlite	Instructs conversion to be from a CSV file to SQLite format. With this option, one needs to also specify the -dst <resulting database> parameter to tell what new database base to create. If a previous database was created using csvdx , then that can be specified as well and the data will be added to the database.
-split_into_files	This is specific to TZWorks tools and their output. Specifically, some artifacts, such as the eventlog and registry hives, will have differing header fields that are specific to differing event IDs or type of registry artifact. This option will take the single CSV file containing these differing artifacts and separate each unique one into a separate CSV

	file. With this option, one needs to also specify the -dst <folder> parameter to tell which folder to put the new CSV files.
-pipe	This option allows one to pipe in multiple CSV files from standard input while storing the CSV artifact data into a SQLite database.
-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 or -enumdir options. The syntax is -filter <"*.ext *partialname* ..."> . The wildcard character '*' is restricted to either before the name or after the name.
-quiet	This option is tells csvdx not to display progress status during a run
-flat	Used in conjunction with the -csv_to_json option to have 1 line per entry, versus the normal JSON format that has multiple lines per entry.
-no_header_info	Tells tool that you are not parsing a CSV file with a TZWorks banner, so it will not look for it.

3.2 CSV Specific Sub-Options

These options are only used to modify the properties of the CSV format. At a minimum the **-src <source file>** must be specified. If no **-dst <resulting file>** is specified, the resulting output will be sent to standard output (eg. the screen), which can be redirected into another tool or file. If using the **-pipe** option, then the **-dst <resulting directory>** must be specified.

Option	Description
-no_whitespace	This option will remove all white space between the field value and the CSV separator.
-pair_fields	This option will pair two separate fields into one. Format is: -pair_fields "1:2, 3:4 ..." . The 1:2 notation means combine column 2 into column 1. The 3:4 notation means combine column 4 into column 3. This is useful when desired to combine date field and time field into one field containing both date and time.

-align_cols	This option will space out the delimited fields and try to align the columns. This is useful to view the CSV output in a more readable manner when using text viewer such as notepad.
-change_csv_separator	This option is to modify the existing CSV delimiter to some other delimiter. Format is: -change_csv_separator "I" .
-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.

3.3 SQLite Specific Sub-Options

If one is not that familiar with SQLite or does not want to use SQL queries to analyze the resulting SQLite database, we built in a couple of shortcut options. The first two options are to enumerate the records of the two baseline tables (*ref* and *metadata*). The third is for extraction of the data in the artifact tables (and the *metadata* table) to generate discrete CSV files for each artifact type.

Option	Description
-ref_records	This option is used to read the resulting <i>ref</i> table and display the records. The format is: -src <SQLite db> -ref_records
-metadata_records	This option is used to read the resulting <i>metadata</i> table and display the records. The format is: -src <SQLite db> -metadata_records
-artifact_tables	This option is used to read the resulting SQLite database and output each artifact table into unique CSV files specified at the specified directory. The format is: -src <SQLite db> -dst <directory to store files> -artifact_tables .
-sqlite_to_csv	This option is used to read a specific table and output the rows to a specified output file in CSV format. This option assumes the data can be read as ASCII data (not to be used for binary data). The format is: -src <SQLite db> -dst <csv results> -sqlite_to_csv <table_name_to_output> .

4 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.

4.1 *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 indicate 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.

5 References

1. TZWorks tools and the CSV outputs they produce
2. JSON Organization: <http://json.org/>
3. JSON Data Interchange Format, ECMA-404, <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>
4. W3C HTML Specification, <http://www.w3.org/TR/html/>
5. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].
6. SQLite documentation [<http://www.sqlite.org>].