

TZWorks® Event Log MessageTables Offline (*elmo*) Users Guide



Abstract

elmo is a standalone, command-line tool that can traverse a system volume (either live or archived) and pull the various Windows event log MessageTables into an SQLite database. This database can then be queried later in an offline manner to find messages that equate to specific event identifiers.

Copyright © TZWorks LLC

www.tzworks.com

Contact Info: info@tzworks.com

Document applies to v0.46 of ***elmo***

Updated: Apr 25, 2025

Table of Contents

1	Introduction	2
2	Background Information	3
2.1	Separation of Source Code from Language Specific Resources.....	3
2.2	How Language Names Relate Locale Code Identifiers (LCIDs).....	4
2.3	Message String Arguments	5
2.4	Where are the MESSAGETABLE's.....	5
3	How to Use <i>elmo</i>	7
3.1	Database Creation (or Database Update).....	7
3.1.1	Create from a Live Volume	7
3.1.2	Create from a captured System Mounted Volume.....	8
3.1.3	Create from an Off-Line (unmounted) Image	9
3.1.4	Create from discrete files.....	9
3.2	Querying the database using <i>elmo</i>	9
3.2.1	Table Enumeration.....	9
3.2.2	Event ID Query	10
3.3	Using <i>elmo</i> with <i>evtwalk</i> CSV Data	12
4	SQLite Notes.....	13
4.1	SQLite Dependencies	13
4.2	Database Schema used by <i>elmo</i>	13
4.3	Providers	14
4.4	Mapping of an Event to <i>elmo</i> Database.....	15
4.5	Handling Multiple Languages.....	16
5	Available Options	17
6	Authentication and the License File.....	19
7	References	20

TZWorks® Event Log Message Tables Offline (*elmo*) Users Guide

Copyright © TZWorks LLC

Webpage: http://www.tzworks.com/prototype_page.php?proto_id=35

Contact Information: info@tzworks.com

1 Introduction

elmo is a prototype command line utility to assist the analyst in pulling message table data from providers with the object of integrating these messages to events that are logged in the Windows event log.

The Windows event log conserves space using a number of mechanisms. One way is to reference the *provider* for each event along with unique event information in the log and store the more common information in a resource binary. The term *provider*, as used here, is the source of the event that was generated and is recorded in the event log. This can be one of the running services, drivers, or applications. Reconstruction of the complete message for an event that is logged therefore requires one to pull the common message strings from the resource that houses the provider's information. **elmo** is a utility to help the analyst do this.

One can examine the various providers for the event log by looking at the *System* hive in the Windows registry, and for newer operating systems, the additional information in the *Software* hive. Each provider, in turn, points to one or more PE (portable executable) files that contain an embedded table of messages (referred to by Microsoft as a MESSAGETABLE). Within the table of messages, each item in the table equates to an event identifier which is referenced when logging events in the event log. In this way, boiler plate phrases or sentences can be offloaded to the MESSAGETABLE resources, and only the unique values that populate the data in the message need to be stored in the log itself.

These dependencies are integrated in a seamless manner when analyzing event logs using Microsoft tools and when on the same target machine that the log file was on. Doing it offline however, or using a different tool to parse the event log, can be problematic, since understanding the dependencies can be error prone and most of the techniques to extract the entry from the appropriate message table tend to be manual. To complicate the process further, PE files with embedded MESSAGETABLE's can change from one operating system to the next. As an example, the MESSAGETABLE from the PE file *msaudite.dll*, which is the main one for Security event auditing, differs from WinXP to Win7 to Win8. While some of the events IDs match the messages across the operating systems, some do not. This is also the case when considering different language types. Integrating all these MESSAGETABLES across differing operating systems is thus a difficult task if tried to do manually.

When architecting **elmo**, our objective was to have a self-contained way to pull all the appropriate message tables from a target computer and archive this data into a database that allows for easy retrieval. SQLite was chosen as the database engine, since it is lightweight, portable and ubiquitous across various operating systems. To make **elmo** handle a number of different use-cases, we designed

elmo to pull data from: (a) a live computer, (b) 'dd' image of a system volume, (c) a mounted system volume image that was extracted from another box, or (d) discrete files pulled from a target box. For some of the database creation use-cases, **elmo** needs to run at administrative privileges. Once the database is created, **elmo** can be used for event log message re-creation and can be run at the typical user privileges. While **elmo** doesn't address all the issues for integrating events to message tables, it is a good start.

2 Background Information

Since message tables are a key part of event message reconstruction, we added this brief section to provide background information on: what are the message table components, where they come from and how these components relate to each other. Arming oneself with this type of information will help one understand the information **elmo** exposes and what it means.

2.1 Separation of Source Code from Language Specific Resources

When dealing with message tables (MESSAGETABLE's) one needs to be familiar with the term MUI, which is short for Multilingual User Interface. From a worldwide perspective, there are over 6900 known living languages in use. MSDN documents that the purpose of MUI is to "separate the storage localization resources from application source code, so as to be able to architect any multilingual application as a combination of language-neutral core binary and as set of language-specific localized resource files". Event logs use this concept as well.

MESSAGETABLE's are defined using a message compiler (.mc) files and are compiled into resource files using the Microsoft message compiler tool, mc.exe (that is distributed with Visual Studio as well as the Windows Software Development Kit). The format of the message table is designed so that multilingual error messages are easier to interpret.

Language names can be specified as either language names or Locale Code Identifiers (LCIDs). Below are example of English and German language names and their companion LCID's.

English: en-US or LCID 0x0409

German: de-DE or LCID 0x0407

Give the above, one can see that the LCID 0x0409 equates to the US version of English language name (en-US). Even though the language name and LCID are different entities, in this document they are used interchangeably since they directly relate to one another.

2.2 How Language Names Relate Locale Code Identifiers (LCIDs)

Locale code identifiers are combination of a primary language ID (10 bits) and a secondary language ID (6 bits) to form a 16 bit word. Below is a subset list of the primary language ID's from 0x01 to 0x19.

Primary ID	Abbreviation	Description
0x01	ar	Arabic
0x02	bg	Bulgarian
0x03	ca	Catalan
0x04	zh	Chinese
0x05	cs	Czech
0x06	da	Danish
0x07	de	German
0x08	el	Greek
0x09	en	English
0x0a	es	Spanish
0x0b	fi	Finnish
0x0c	fr	French

Depending on the primary language ID, the secondary language ID is a subset of the primary. For example, for English one would use a primary ID of 0x09; below are some secondary IDs that can be used for English.

Secondary ID	Abbreviation	Description	Final LCID
0x04	en-us	English – United States	0x0409
0x08	en-gb	English – United Kingdom	0x0809
0x0c	en-au	English – Australia	0x0c09
0x14	en-nz	English – New Zealand	0x1409
0x18	en-ie	English – Ireland	0x1809
0x1c	en-za	English – South Africa	0x1c09
0x24	en-cb	English – Caribbean	0x2409
0x28	en-bz	English – Belize	0x2809
0x2c	en-tt	English - Trinidad	0x2c09
0x34	en-ph	English - Philippines	0x3409

The incrementing of the secondary ID looks strange, but it is actually in succession; it is just an artifact of the primary ID using the first 10 bits of the 16 bit word.

2.3 Message String Arguments

For event logs to use pre-canned messages stored in provider resources and create unique messages for each event ID, they make use of *string arguments* within the message resource. Each argument is preceded by a percent '%' character. The argument is another character that is used for an action. The table below is taken from the Microsoft documentation on MESSAGETABLE string arguments and contains most of the arguments that one will see in the provider message resource.

The arguments that have a number after the percent character are place holders that directly equate to the event field that has the same index number. In this way, each event can uniquely put the proper data in the specified position when recreating the entire message for that event.

More information on arguments and their meaning can be found at this URL [ref: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms679351\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679351(v=vs.85).aspx)]:

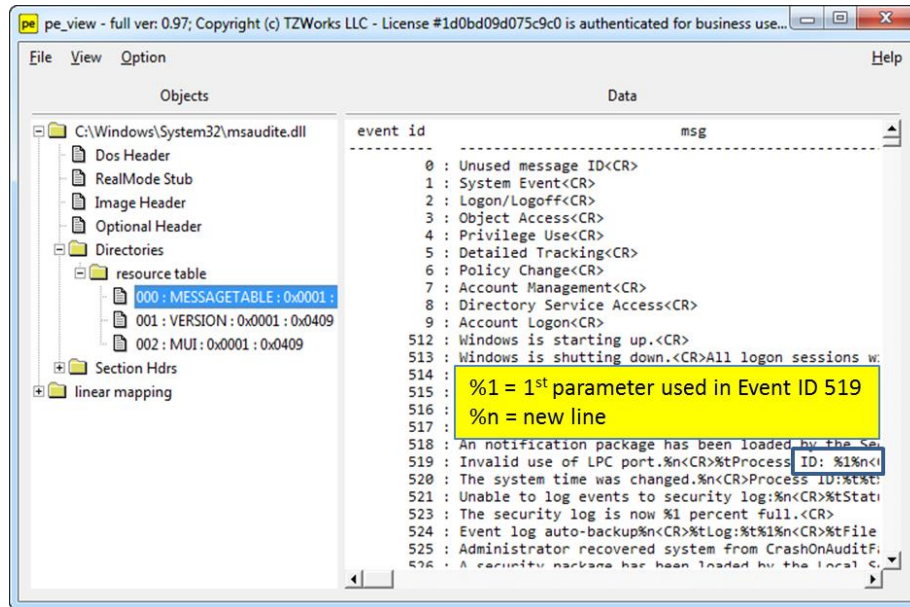
Value	Description
%%	A single percent
%<space>	A single period space
%.	A single period
%!	A single percent exclamation point
%n	A hard line break
%r	A hard carriage return without a trailing newline character
%t	A single tab
%0	Terminates a message text line without a trailing new line character
%1 - %99	Argument place holder
%1!format string! - %99!format string!	Argument place holder with format specifier that is similar to the printf formatting.

2.4 Where are the MESSAGETABLE's

It is sometimes useful to see exactly where these MESSAGETABLE's come from.

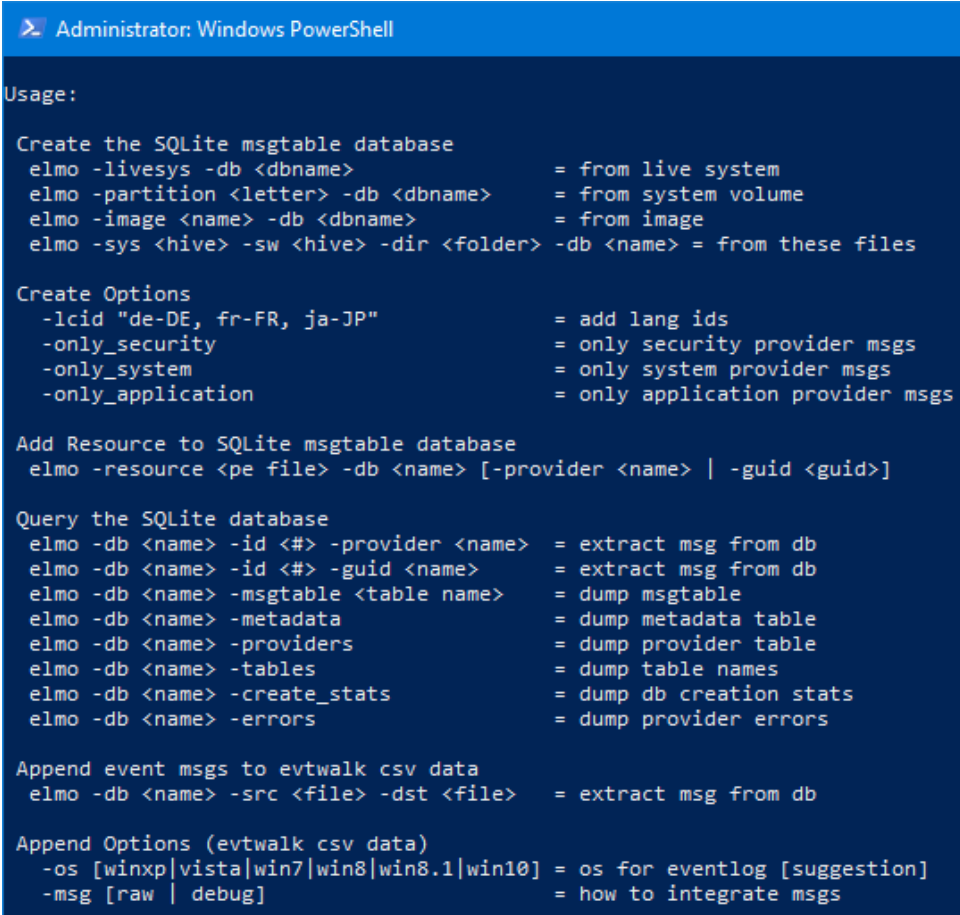
Below is what a MESSAGETABLE looks like for the Security Provider "*msaudite.dll*". Highlighted is one of the messages in the table equating to event ID 519. One can repeat this, by opening up "*msaudite.dll*" from the System32 directory using **pe_view** from *TZWorks*. The MESSAGETABLE is embedded in the resource section of the PE file.

This is the data that **elmo** extracts and catalogs for easy retrieval during its database creation phase.



3 How to Use *elmo*

The screen shot below shows all the options available. The options are grouped into categories that equate to three basic functions: (a) create a SQLite database of MESSAGETABLE's, (b) query the database to find specific data within the database, and (c) analyze the CSV data produced by the TZWorks® tool *evtwalk* and annotate the CSV data with message data.



```
Administrator: Windows PowerShell

Usage:

Create the SQLite msgtable database
elmo -livesys -db <dbname>           = from live system
elmo -partition <letter> -db <dbname> = from system volume
elmo -image <name> -db <dbname>       = from image
elmo -sys <hive> -sw <hive> -dir <folder> -db <name> = from these files

Create Options
-lcid "de-DE, fr-FR, ja-JP"          = add lang ids
-only_security                       = only security provider msgs
-only_system                         = only system provider msgs
-only_application                    = only application provider msgs

Add Resource to SQLite msgtable database
elmo -resource <pe file> -db <name> [-provider <name> | -guid <guid>]

Query the SQLite database
elmo -db <name> -id <#> -provider <name> = extract msg from db
elmo -db <name> -id <#> -guid <name>    = extract msg from db
elmo -db <name> -msgtable <table name>  = dump msgtable
elmo -db <name> -metadata               = dump metadata table
elmo -db <name> -providers              = dump provider table
elmo -db <name> -tables                 = dump table names
elmo -db <name> -create_stats           = dump db creation stats
elmo -db <name> -errors                 = dump provider errors

Append event msgs to evtwalk csv data
elmo -db <name> -src <file> -dst <file> = extract msg from db

Append Options (evtwalk csv data)
-os [winxp|vista|win7|win8|win8.1|win10] = os for eventlog [suggestion]
-msg [raw | debug]                     = how to integrate msgs
```

3.1 Database Creation (or Database Update)

Shown in the menu above, there are four ways to create or update an *elmo* database with new MESSAGETABLE data. The database that is created or updated is formatted as SQLite. The sections below describe each of these ways. For Linux and OS-X versions of *elmo*, this is reduced to two ways ('dd' image and discrete files).

3.1.1 Create from a Live Volume

Creating a database from a live volume requires one to run *elmo* with administrator privileges. This option is only available when running on Windows operating systems. To invoke this command one uses the *-livesys* switch and specifies that name of the database to create via the *-db <path/dbname>*.

The resulting database that is operating on can be a new database (in which case it is created) or an existing database. If specifying an existing database, **elmo** will only add new MESSAGETABLE entries to the database. The term 'new' is defined here, in the sense that the hash of the MESSAGETABLE doesn't already exist in the database.

When creating new databases, it is recommended using a name that describes the target machine operating system and the default language used. Creating a separate **elmo** database for each Windows OS and/or language ID is the most reliable way to ensure that the proper message is matched with the event ID. The importance of a good naming convention will become more apparent later when trying to use the database to correlate messages to event IDs.

For example, on a Windows 7 box, where you think the default language ID is **en-US**, one might use:

```
elmo64 -livesys -db win7_en_us.db
```

Be aware, that even though the name includes *en_us*, what **elmo** does internally is look at the default language ID that is on the operating system and targets it. So if it happens to be *German*, then it will target MESSAGETABLE resources that are *German* versus *English*. **elmo** does, however, include an option to force it to look at other language identifiers (LCIDs), via the **-lcid** option. One can specify any LCID one wants via this option and **elmo** will scan the system volume for MESSAGETABLE resources with those LCIDs, in addition to the default LCID.

Another factor to consider when using **elmo** to create databases is that **elmo** will scan all the *providers* identified by the *System* hive (and depending on OS version, the *Software* hive as well) of the Windows registry. From each *provider* found, it will then parse each of the PE resources that were identified. This can amount to a lot of data in one database. If one only wants to target a specific category of providers, there are a couple of optional switches: **-only_security**, **-only_system**, and **-only_application**, to target the respective providers from security, system or applications. While there are other providers, these are the main ones. Suffice it say, if one wants all to pull all the providers MESSAGETABLE resources then invoke **elmo** using the default behavior by not specifying any of these optional switches.

3.1.2 Create from a captured System Mounted Volume

If one captures an image of a Windows system volume as a file and it can be mounted on a separate workstation as a separate drive letter, then **elmo** can target this mounted volume. The proviso is that the mounted system volume is mounted as a 'block device' where the entire filesystem is exposed without additional aliases.

This option is similar to the previous one with a live system volume, but just targets a separated system volume mounted as another drive letter. One uses the option: **-partition <drive letter>** to invoke this. All the same optional switches discussed above apply here.

This option is only available to Windows and requires one to run **elmo** with administrator privileges.

3.1.3 Create from an Off-Line (unmounted) Image

If one captures an image of a Windows system volume as a single 'dd' file, one can point this *elmo* to this file. This option does not require any special permissions when running *elmo*, however it does require the file to be a monolithic 'dd' type image of a Windows system volume. What this means is the 'dd' image consists of one file, and not multiple files that are parts of one image. This mode can be run from any compiled version of *elmo*, such as OS-X or Linux. The same optional switches discussed above apply here.

To invoke this option, use the syntax: **-image <path/filename of the 'dd' image>**. The image option is similar to the live volume in the sense that *elmo* will locate the proper registry hives, parse them, and based on the registry information, locate the PE resources, extract the MESSAGETABLE data and build a database that can be used offline.

3.1.4 Create from discrete files

Lastly one can tell *elmo* to create (or update) a database directly from separate files that were extracted from a target box. The necessary files would include: (a) Windows system and software registry hives and (b) the PE files that contain the MESSAGETABLE resources. One uses the options **-sys <system hive>** and **-sw <software hive>** to specify the system and software hives, respectively. The **-dir <directory of PE resources>** option specifies the directory where to locate the PE resources that are referenced by the data in the system and software hives.

Using this option does not require any special permissions when running *elmo*. Also, this option can be run on Linux and OS-X. There are, however, some limitations when using this mode. The biggest limitation is making available all the PE resource files needed so that a complete listing of event log message tables can be cataloged within the database that is created or updated. Further, one must ensure the PE resource files included in the directory match those specified as providers in the system and software hives. For this reasons, it is best (and less problematic) to use one of the previous options which rely on the Windows system volume being present during the database creation.

3.2 Querying the database using *elmo*

One can use SQLite tools to query the database, or one can use the built in *elmo* commands to enumerate any of the tables, their contents, or to search for matching messages given an event ID and its associated provider.

3.2.1 Table Enumeration

For general table enumeration functions, one can use the appropriate option to enumerate all the entries of a desired table. For example, the options **-tables**, **-providers**, and **-metadata** refer to the respective tables: *ref*, *_providers* and *metadata*. The option: **-msgtable < table name >** refers to the *PE*

resource tables, where the table name is the name of the PE file. If there are questions about how the data in the database was gathered one can use the **-create_stats** to enumerate the *_genesis* table, which will include, among other things, the command line used during database creation. Multiple entries imply that multiple target operating systems were used to create the database. If there were some PE resource entries that you thought should have been parsed but were not, use the **-errors** option to enumerate all the problem entries that were discovered and the reason why *elmo* thought it was a problem.

Finally, in the category of table enumeration, if one instead wants to *view* any of the table contents in a *graphical* sense, the SQLite browser from SQLite.org works well.

3.2.2 Event ID Query

For spot event ID queries, one can input an event ID and provider name (or GUID) and get the associated event message. To do this, one uses the option: **-id <#> -provider <name>**. In some cases, more than one message will be returned. This is because an event ID may have multiple qualifiers that are embedded into the high order bits of the raw event ID, where the lower 16 bits are the identifier itself. Another reason that one may get multiple entries is if the database was created using PE resources from multiple Windows operating systems. This usually causes overlap between event IDs.

Below is an example where we created a single database from both Win7 and WinXP images and queried an event identifier that 615 from the Security provider. The PE resource file used by the Security Provider for both Win7 and WinXP is *msaudite.dll*. While most PE resources that span multiple operating systems have similar messages, some do not, like that in the example below.

```
"cmdline: elmo64 -db test.db -id 615 -provider security"

Event ID:      0x00000267
Event code:    615
Severity:      Success
LogType:       security
Provider:      security
SrcMsgFile:    msaudite.dll
TypeMsgFile:   Category; Event
LCID:          en-US
Min OS:        Win7 or Win2008/R2
Message:       %1

Event ID:      0x00000267
Event code:    615
Severity:      Success
LogType:       security
Provider:      security
SrcMsgFile:    msaudite.dll
TypeMsgFile:   Category; Event
LCID:          en-US
Min OS:        WinXP
Message:       IPsec Services: %t%1%n
```

Everything is the same, except the OS version and event message

Win7

WinXP

In this case the first message is just the characters "%1" and the second is the message "IPsec Services: %t%1%n". The "%1" is the way Microsoft defines a placeholder for a string argument. The data for the

string argument is taken from the parsed event log and substituted into the message. The other variables, %t and %n are for a tab character and line feed character, respectively. So, while this example has a very short message, it was shown to enlighten the user that the combination of *Provider* and *Event ID* do not necessarily yield unique messages. As operating systems are enhanced, the names of many of the system PE resources stay the same, but their respective message tables may or may not change.

To help one understand the data that is outputted by *elmo*, the definition of the above fields is shown in the table below:

Field	Definition / option	Source of data
Event ID	Unfiltered 4 bytes of the event ID	Message table
Event code	Lower 2 bytes of the event ID	Derived from Event ID
Severity	High 2 bits of the event ID. Options are: <i>00 - Success,</i> <i>01 - Informational,</i> <i>10 - Warning,</i> <i>11 - Error</i>	Derived from Event ID
LogType	Which type of event log this event applies to. Common ones are: <i>Security, System and Application.</i>	Parent key of the provider of the eventlog in registry hive
Provider	Name of the provider that pointed to this PE file containing the message table	Taken directly from Providers in System and/or Software hives
SrcMsgFile	PE file containing the message table	Usually in System32 directory
TypeMsgFile	Relates to type of MESSAGETABLE. Options are: <i>1 - Event (EventMessageFile),</i> <i>2 - Category (CategoryMessageFile),</i> <i>3 - Parameter (ParameterMessageFile),</i> <i>4 - Guid (GUIDMessageFile)</i>	Taken directly from Providers identified in System and/or Software hives
LCID	Language Code Identifier	MUI name or PE resource internals
Min OS	Minimum Window OS that this PE resource is applicable for	Directly from PE resource internals
Message	Raw Message table entry associated with Event ID	Directly from PE resource internals

For the *TypeMsgFile* where multiple types are shown, this is because multiple types were either specified in the registry hive or because different providers specified different types for the same PE resource.

One can also query using a provider GUID, which is shown below. Notice, that there is no filtering in the output, so duplicate entries are displayed. We tried to annotate the reasons for each duplicate entry.

```
"cmdline: elmo64 -db test.db -guid {fc65ddd8-d6ef-4962-83d5-6e5cfe9ce148} -id 1"

Event ID: 0x50000001
Event code: 1
Severity: Informational
LogType: security ← Targets Security Eventlog
Provider: microsoft-windows-eventlog
SrcMsgFile: wevtsvc.dll
TypeMsgFile: Event
LCID: en-US
Min OS: Win7 or Win2008/R2
Message: Critical

Event ID: 0x50000001
Event code: 1
Severity: Informational
LogType: system ← Targets System Eventlog
Provider: microsoft-windows-eventlog
SrcMsgFile: wevtsvc.dll
TypeMsgFile: Event
LCID: en-US
Min OS: Win7 or Win2008/R2
Message: Critical

Event ID: 0x90000001 } ← Then there are entries with the
Event code: 1 same event code, but different
Severity: Warning event qualifiers
LogType: security
Provider: microsoft-windows-eventlog
SrcMsgFile: wevtsvc.dll
TypeMsgFile: Event
LCID: en-US
Min OS: Win7 or Win2008/R2
Message: Microsoft-Windows-Eventlog

Event ID: 0x90000001
Event code: 1
Severity: Warning
```

3.3 Using *elmo* with *evtwalk* CSV Data

evtwalk is a TZWorks command line tool for parsing Windows event logs and outputting the results in a CSV fashion. While **evtwalk** pulls all the data available from an event log, it doesn't try to locate and extract the message table from the PE resource. To fill that need, we therefore added the functionality into **elmo**. So, if one has the updated version of **evtwalk** (v0.30 or later), one can take the CSV output from **evtwalk** and pass it into **elmo**. **elmo** in turn will analyze the CSV data and extract certain fields (like the provider and event ID for each record) and look up an associated message table. After this, **elmo** will perform argument substitution by taking the parameter data from the CSV file and generate a message which then gets appended to the CSV data. To ensure there is no corruption with the original CSV data, a new CSV file is created, which is identical to the original **evtwalk** CSV file inputted, but adds a couple of extra fields to each record. These fields include: (a) message translation and (b) task category translation.

This option should be considered experimental, since a number of conditions need to be satisfied for the above process to work. Firstly, it assumes that resulting CSV file produced by **evtwalk** has all the fields needed by **elmo** populated. Basically this means that if you run **evtwalk** in a mode that does not filter any fields, then it should work. If filtering is used and the required fields needed by **elmo** are filtered out, then the results will be unknown. Second, the CSV file needs to have the required field header

names and argument syntax for *elmo* to extract the proper fields during its analysis. If one is using the latest version of *evtwalk* (v0.31 or greater), then this latter requirement is satisfied.

4 SQLite Notes

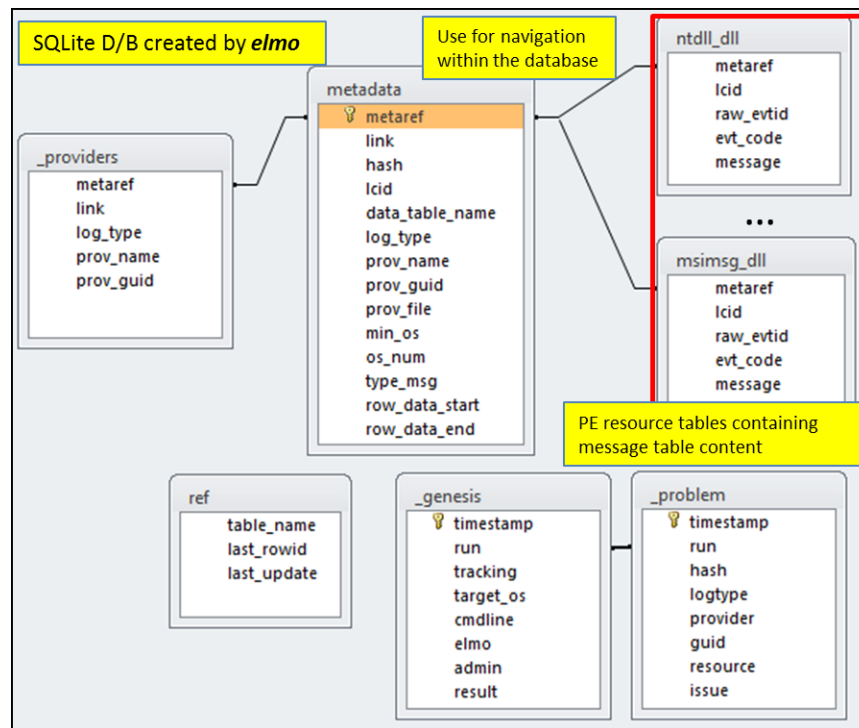
4.1 SQLite Dependencies

elmo makes use 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. Prior versions to v0.24 required a separate shared SQLite library for the tool to run. Starting with v0.24, we have compiled in the static library into *elmo*, so everything is self-contained.

4.2 Database Schema used by *elmo*

If one is interested in the internals of the database that gets created from *elmo*, below is a diagram of the database schema used. There are a few tables to track housekeeping items (tables: *ref*, *_providers* and *metadata*). There are also two tables (*_genesis* and *_problem*) to record the history of what commands were used in database creation and any problems encountered during database creation. These latter two are only used to help diagnose problems. The other set of tables uses a generic table structure to record MESSAGETABLE data of PE resources. These last set consist of one table per unique PE resource.

The table used for navigation and summary information is the *metadata* table. It in combination with the PE resource MESSAGETABLE tables allows *elmo* to quickly pull the candidate messages given an event identifier, provider name and an optional language identifier.



Given the general schema above, one can see the basic relationships for each of the tables. *elmo* has a few built-in options to dump any of the above tables. This is useful if the user is not familiar with SQL queries or needs to look something up quickly. The options for table enumeration are discussed in the section on “Table Enumeration”.

4.3 Providers

In the discussion, the term “Provider” will be used to pinpoint where the event ID references. In some references the name *Publisher* is used instead. For the purposes of *elmo*, the name *Provider* is used for both.

Providers were originally listed in the System hive of the Windows registry, specifically the subkeys located here: `HKLM\SYSTEM\CurrentControlSet\services\eventlog\<event log names>\<provider names>`. So for Window XP and later operating system versions, this location still has a list of *Providers*. With the later versions, a *Provider* name can also have a *GUID* (Globally Unique Identifier) associated with the name, as well as have more details in the Software hive. Specifically, if one looks at the subkeys: `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\<GUID>`, there contains a list of *Provider GUIDs* associated with *Provider Names*.

In the schema shown above, the “_providers” table will contain both the *Provider Name* and *Provider GUID* (if available). This allows one to query an event ID with either *Provider Name* and/or *GUID*.

Related to the *Provider*, is the *log_type* field. This entry specifies an event log name that can contain entries by a *Provider*. In some cases one can see one Provider has entries in multiple event log types.

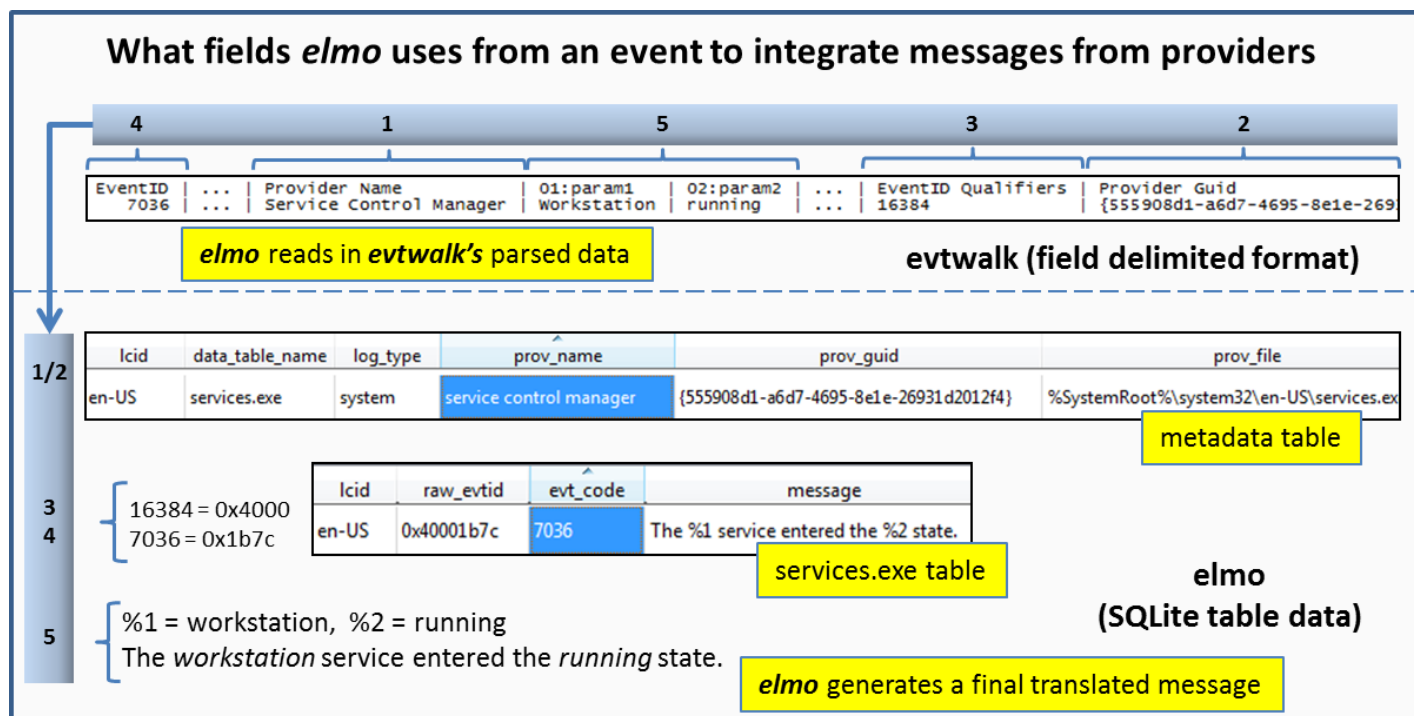
Familiar *log_types* are event logs named: “Security”, “System”, and “Application”. There are many other *log_types* for the new versions of Windows.

4.4 Mapping of an Event to *elmo* Database

When needing to check the results of *elmo* manually, one needs to understand how to map a specific event to a final translated message. To do this accurately, one needs to understand the fields used in the event and how they are matched to records in specific SQLite tables.

As an example, below is Event ID 7036 rendered from the TZWorks® tool *evtwalk*. Highlighted with numbers are the locations of data used by *elmo* to perform automatic lookup and message translation.

The key fields are: (a) Provider name and GUID (shown as 1 and 2 below), (b) Event ID and Qualifier (shown as 3 and 4 below), and (c) any string arguments (shown as 5). The first thing *elmo* looks at is the Provider name and/or GUID and looks up the Provider details in the *metadata table*. If a record is found, *elmo* then finds which PE resource message table was referenced by the Provider. With the PE resource message table, *elmo* then looks for a record with the target event ID and qualifier. If this is located, then the message is extracted. If there are any string arguments that need to be populated with event data, those fields are searched in the event, and if found, extracted and substituted into the string arguments to make a completed message for that event.



4.5 Handling Multiple Languages

The database created by *elmo* internally stores the associated language identifiers for each message table that is extracted. This has a couple of benefits. First it allows multiple languages for the same provider and event ID to co-exist within the database. Second it allows one to query the database and specify which language identifier to use.

To create a database that has message tables with multiple languages, one has a couple of options: (a) the first use-case is during the creation of the initial database, pass in the option **-lcid <language ID's delimited by commas>**. What this tells *elmo* to do is to first look at the default language (from the target box) and then look at any language ID's specified as well. So if one wanted English, French and the German language message tables, one could specify **-lcid "en-US, fr-FR, de-DE"**. The resulting database that is created will be based on the ability of *elmo* to find all the PE resources of the language types that were specified. (b) The second use-case is to run *elmo* with the database creation option for each target box passing in the same database for each one. *elmo*, in this case, will merge new message table data into the database for each run. If the target machines had operating systems with differing default language packs, then the resulting database at the end would have a combination of the message tables for each of those default languages.

Below is an example of doing this on English, French and German. Shown below are some of the fields for the provider "Microsoft-Windows-FMS" in the metadata table that was created. This particular provider points to the PE resource file fms.dll, located at %systemroot%\system32\<lang id>\fms.dll.mui. Each version of the message table resource is stored at the respective <lang id> subdirectory.

lcid	data_table_name	prov_name	prov_guid	prov_file
en-US	fms.dll	microsoft-windows-fms	{dea07764-0790-44de-b9c4-49677b17174f}	%SystemRoot%\system32\en-US\fms.dll.mui
de-DE	fms.dll	microsoft-windows-fms	{dea07764-0790-44de-b9c4-49677b17174f}	%SystemRoot%\system32\de-DE\fms.dll.mui
fr-FR	fms.dll	microsoft-windows-fms	{dea07764-0790-44de-b9c4-49677b17174f}	%SystemRoot%\system32\fr-FR\fms.dll.mui

When looking at the 'fms.dll' data table that was created, one would see entries for each LCID that extracted, which in this case is *en-US*, *de-DE*, and *fr-FR*. For each of these LCID's there is an event code and its associated message for the language.

	metaref	lcid	raw_evtid	evt_code	message
1	1308985716958...	en-US	0x30000000	0	Info
2	1308985723377...	de-DE	0x30000000	0	Info
3	1308985723382...	fr-FR	0x30000000	0	Informations
4	1308985716958...	en-US	0x30000001	1	Start
5	1308985716958...	en-US	0x90000001	1	Microsoft-Windows-FMS
6	1308985723377...	de-DE	0x30000001	1	Starten

Data Table
fms.dll

5 Available Options

Option	Description
-livesys	Create a database from the system volume <i>elmo</i> is being run on. The format is: -livesys -db <resulting db>
-partition	Create a database from a mounted image from some offline system volume. The format is: -partition <drive letter containing mounted system volume> -db <resulting db>
-image	Create a database from an image of a system volume. The image needs to be a monolithic file in 'dd' format. The format is: -image <file with dd image> -db <resulting db>
-sys -sw -dir	Create a database from this specific System registry hive and/or Software hive. Use the PE resources identified in directory specified. The format is: -sys <system hive> -sw <software hive> -dir <system32 directory> -db <resulting db>
-id -provider	Query the message that matches this event ID and provider. The format is: -id <event ID> -provider <name of provider> -db <db to query>
-msgtable	Dump the table data for a specific MESSAGETABLE resource. The format is: -msgtable <table name> -db <db to query>
-metadata	Dump the metadata table. The format is: -metadata -db <db to query>

-providers	Dump the providers table. The format is: -providers -db <db to query>
-tables	Dump the tables used in the <i>elmo</i> database. The format is: -tables -db <db to query>
-create_stats	Dump the creation table stats. The format is: -create_stats -db <db to query>
-errors	Dump the provider errors. The format is: -errors -db <db to query>
-lcid	Include the following language IDs in the MESSAGETABLE resource extraction during the database create option. The format is: -lcid "de-DE, fr-FR, ja-JP, en-US, ..." [any of the database creation options]
-only_security	Only pull security providers during the extraction of MESSAGETABLE resources during the database create option. The format is: -only_security [any of the database creation options]
-only_system	Only pull system providers during the extraction of MESSAGETABLE resources during the database create option. The format is: -only_system [any of the database creation options]
-only_application	Only pull application providers during the extraction of MESSAGETABLE resources during the database create option. The format is: -only_application [any of the database creation options]
-src -dst	The -src option specifies the CSV file containing the output of evtwalk parsed data. The -dst option specifies where you want the resulting data to put. -src <evtwalk CSV file> -dst <results file> -db <db to use for message tables>
-os	Target the following Windows operating system event log message tables (if possible). The format is: -os [winxp/vista/win7/win8/win8.1/win10]
-msg	Option for -src / -dst for parsing <i>evtwalk</i> CSV output, to tell how you want the messages to be integrated into the final output. raw = show raw message with no parameter substitution; debug = provide parameter substitution along with original argument string. Default mode does parameter substitution without the argument string. If desiring default mode, don't use this option. The format is

	<i>-msg [raw debug]</i>
--	----------------------------------

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.

7 References

1. Microsoft Portable Executable and Common Object File Format [Specification](#).
2. An In-Depth Look into the Win32 Portable Executable File Format, by Matt Pietrik, MSDN Magazine.
3. Wikipedia, the free encyclopedia. [PE format](#)
4. String Message Arguments. [https://msdn.microsoft.com/en-us/library/windows/desktop/ms679351\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679351(v=vs.85).aspx)
5. Detailed explanation of the message text file:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tools/mc_77lf.asp
6. Parsing Event log data with theTZWorks® evtwalk tool.
https://tzworks.com/prototype_page.php?proto_id=25
7. Viewing of MESSAGETABLE data in PE Resource with the TZWorks® pe_view tool.
https://tzworks.com/prototype_page.php?proto_id=7
8. SQLite library statically linked into tool [Amalgamation of many separate C source files from SQLite version 3.32.3].
9. SQLite documentation [<http://www.sqlite.org>].