Event Log Explorer

Scripting Reference

Table of contents

Introduction

Event Log Explorer comes with scripting support provided by FastScript/PascalScript (https://www.fast-report.com/en/product/fast-script/). Scripting support is available in Forensic and Enterprise Editions of Event Log Explorer.
Pascal script is based on Delphi Object Pascal language, but with some limitations:
- No type declarations (records, classes) in the script;
- No records;
- No sets (but you can use 'IN' operator - "a in ['a'..'c','d']");
- No shortstrings;
- No GOTO statement;
- No Generics;
- No Anonymous Methods.

Data types

Internally FastScript operates with the Variant type and is based on it. Nevertheless, you can use the following predetermined types in your scripts:

Integer types: BYTE, WORD, Integer, Longint, Cardinal, DWORD, TColor
Boolean types : Boolean
Floating point types: Real, Single, Double, Extended
Date and time types (mapped to Extended): TDate, TTime, TDateTime
Character type: Char
String type : String
Variant type: Variant, Pointer
Array type: Array

Global procedures and functions

There is a rich set of standard FastScript functions which can be used in a script. These functions are mapped to Delphi Object Pascal functions and you can get more information from the documentation at https://docwiki.embarcadero.com/Libraries/

function **IntToStr**(i: Integer): String;
Converts integer (i) to String.

function **FloatToStr**(e: Extended): String;
Converts float (e) to String.

function **DateToStr**(e: Extended): String;
Converts TDate (e) to String using the local settings for displaying date and time.

function **TimeToStr**(e: Extended): String;
Converts TTime (e) to String using the local settings for displaying date and time.

function **DateTimeToStr**(e: Extended): String;
Converts TDateTime (e) to String using the local settings for displaying date and time.

function **VarToStr**(v: Variant): String;
Converts variant (v) to String.

function **StrToInt**(s: String): Integer;
Converts string (s) to Integer. If s doesn't represent a valid number, this function will raise an exception.
You can use hexadecimal value by adding prefix '$' or '0x' or 'x'. E.g. StrToInt('$A'); StrToInt('0xa').

function **StrToFloat**(s: String): Extended;
Converts string (s) to floating-point value. If s doesn't represent a valid number, this function will raise an exception.

function **StrToDate**(s: String): Extended;
Converts string (s) to a TDate value. If s does not contain a valid date, StrToDate raises an exception.

function **StrToTime**(s: String): Extended;
Converts string (s) to a TTime value. If s does not contain a valid time, StrToTime raises an exception.

function **StrToDateTime**(s: String): Extended;
Converts string (s) to a TDateTime value. If s does not contain a valid date, StrToDateTime raises an exception.

function **Format**(Fmt: String; Args: array): String;
Returns a formatted string assembled from a format string and an array of arguments.

function **FormatFloat**(Fmt: String; Value: Extended): String;
Formats the floating-point value given by Value using the format string given by Fmt

function **FormatDateTime**(Fmt: String; DateTime: TDateTime): String;
Formats the TDateTime value given by DateTime using the format string given by Fmt

function **FormatMaskText**(EditMask: string; Value: string): string;
Returns a string formatted using an edit mask.

function **EncodeDate**(Year, Month, Day: Word): TDateTime;
Returns a TDateTime value that represents a specified Year, Month, and Day. If the specified values represent an invalid date (e.g. Month > 12), this function raises an exception.

procedure **DecodeDate**(Date: TDateTime; var Year, Month, Day: Word);
Returns Year, Month, and Day values for a TDateTime value.

function **EncodeTime**(Hour, Min, Sec, MSec: Word): TDateTime;
Returns a TDateTime value that represents a specified Hour, Min, Sec and MSec. If the specified values represent an invalid time (e.g. Hour > 24), this function raises an exception.

procedure **DecodeTime**(Time: TDateTime; var Hour, Min, Sec, MSec: Word);
Returns hours, minutes, seconds, and milliseconds for a TDateTime value.

function **Date**: TDateTime;
Returns the current date.

function **Time**: TDateTime;
Returns the current time.

function **Now**: TDateTime;
Returns the current date and time.

function **DayOfWeek**(aDate: DateTime): Integer;
Returns the day of the week of the specified date as an integer between 1 and 7, where Sunday is the first day of the week and Saturday is the seventh.

function **IsLeapYear**(Year: Word): Boolean;
Returns True if the specified Year is a leap year and False if the specified Year is not a leap year.

function **DaysInMonth**(nYear, nMonth: Integer): Integer;
Returns the number of days in nMonth of nYear.

function **Length**(s: String): Integer;
Returns the length of String (s) in characters.

function **Copy**(s: String; from, count: Integer): String;
Returns a substring containing Count characters starting at S[From].

function **Pos**(substr, s: String): Integer;
Returns the index of the first occurence of Substr in S

procedure **Delete**(var s: String; from, count: Integer): String;

Removes a substring of Count characters from string S, starting with From.

procedure **Insert**(s: String; var s2: String; pos: Integer): String;
Inserts a substring s into a string S2 beginning at Pos position.

function **Uppercase**(s: String): String;
Returns a copy of a string in uppercase.

function **Lowercase**(s: String): String;
Returns a copy of a string in lowercase.

function **Trim**(s: String): String;
Removes leading and trailing spaces and control characters from a string.

function **NameCase**(s: String): String;
Returns a copy of a string where each word has the first character in uppercase and the other characters in lowercase.

function **CompareText**(s1, s2: String): Integer;
Compares S1 and S2 and returns 0 if they are equal. If S1 is greater than S2, CompareText returns an integer greater than 0. If S1 is less than S2, CompareText returns an integer less than 0. CompareText is not case sensitive.

function **Chr**(i: Integer): Char;
Returns the character with the ordinal value (ASCII value) of the byte-type expression, X.

function **Ord**(ch: Char): Integer;
Returns the ordinal value of the char ch.

procedure **SetLength**(var S: String; L: Integer);
Sets the length L of string S.

function **Round**(e: Extended): Integer;
Rounds floating-point value (e)

function **Trunc**(e: Extended): Integer;
Truncates floating-point value (e) to an integer-type value.

function **Int**(e: Extended): Integer;
Same as Trunc.

function **Frac**(e: Extended): Extended;
Returns the fractional part of a real number.

function **Sqrt**(e: Extended): Extended;
Returns the square root of e.

function **Abs**(e: Extended): Extended;
Returns the absolute value of e.

function **Sin**(e: Extended): Extended;
Calculates the sine of the angle, in radians.

function **Cos**(e: Extended): Extended;
Calculates the cosine of the angle, in radians.

function **Tan**(e: Extended): Extended;
Calculates the tangent of e, in radians.

function **ArcTan**(e: Extended): Extended;
Calculates the arctangent of a given number, in radians.

function **Exp**(X: Extended): Extended;
Returns the exponential of X.

function **Ln**(X: Extended): Extended;
Returns the natural logarithm of X.

function **Pi**: Extended;
Returns Pi.

procedure **Inc**(var i: Integer; incr: Integer = 1);
Increments integer (i) by 1 or by incr.

procedure **Dec**(var i: Integer; decr: Integer = 1);
Decrements integer (i) by 1 or by decr.

procedure **ShowMessage**(Msg: Variant);
Displays message Msg.

function **InputQuery**(Caption, Prompt: string; var Value: string): Boolean;
Displays a dialog box with the given Caption and Prompt message. It asks the user to enter data in a text box on the dialog.
If the user presses OK, the entered data is stored in the Value variable and the function returns True.
If the user cancels the dialog, then the return value is False and any entered data is lost.
See also: InputInt.

procedure **Randomize**;
Initializes the random number generator.

function **Random**: Extended;
Returns a random number within the range 0 <= X < Range.

function **ValidInt**(cInt: String): Boolean;
Indicates if cInt represents a valid integer.

function **ValidFloat**(cFlt: String): Boolean;
Indicates if cFlt represents a valid floating-point value.

function **ValidDate**(cDate: String): Boolean;
Indicates if cDate represents a valid date value.

function **ExtractFilePath**(const FileName: string): string;
Extracts drive and directory parts of filename.

Event Log Explorer adds the following global procedures and functions

function **ExtractFileName**(FileName: string): string;
Extracts name and extension of filename.

function **GetCmdLineParamCount** : Integer;
Returns the number of command-line parameters given to Event Log Explorer when started.

function **GetCmdLineParamStr**(Index : Integer) : String;
Returns the parameter from the command line that corresponds to Index. If Index = 0, this function returns the full path to the executable, e.g. 'C:\Program Files (x86)\Event Log Explorer\elex.exe'. You can use this function to pass parameters to the script.
You can start the program, autorun the script and pass parameters as follows:

```
"C:\Program Files (x86)\Event Log Explorer\elex.exe" /RunScriptConsole
myscript.pas C:\Logs\Security.evtx S-1-5-18
```

In your code you should use GetCmdLineParamStr(3) to get 'C:\Logs\Security.evtx' and GetCmdLineParamStr(4) to get 'S-1-5-18'.

function **GetCmdLine** : String;
Returns the command line with all the parameters.

function **FileExists** (FileName : String) : Boolean;
Indicates if the specified file exists.

function **DirectoryExists** (DirName : String) : Boolean;
Indicates if the specified folder exists.

procedure **GetDirList**(DirMask: String; DirList : TStringList);
Retrieve the list of folders on the disk matching the DirMask.
Example:
GetDirList('C:*', DirList) will retrieve the list of folders in the root of C: disk.

procedure **GetFileList**(FileMask: String; FileList : TStringList);
Retrieve the list of file on the disk matching the FileMask.
Example:
GetDirList('C:\Logs*.evtx', FileList) will retrieve the list of EVTX files in C:\Logs

function **GetTickCount** : DWORD;
Retrieves the number of milliseconds that have elapsed since the system was started. See Windows API function GetTickCount();

This procedures and functions are available only in Script Console. They are not available in custom column formulas:

procedure **DebugOut**(val1, val2....);
Prints parameters to the debug console.

procedure **ClearConsole**;
Clears the debug console output.

procedure **HideConsole**;
Hides the debug console.

procedure **ShowConsole**;
Shows the debug console.

function **GetDescriptionItem**(Descr: String; Level1, Level2 : String) : String;
Retrieves a description item from the formatted description of security and events.
Example:
Consider that event description is

```
Special privileges assigned to new logon.

Subject:
    Security ID:         S-1-5-21-1210403944-1825403922-4215135662-500
    Account Name:        Administrator
    Account Domain:      WIN-DOMAIN1
    Logon ID:            0x000000000003BA74

Privileges:      SeSecurityPrivilege
                 SeTakeOwnershipPrivilege
```

To get the account name, you should call this function as follows:
`GetDescriptionItem(Description, 'Subject', 'Account Name');`

This function works only if the description is formatted with levels and has 'key: value' structure.
It is strongly recommend to use XPath queries to get description details instead of using this function.

function **InputInt**(Caption : String; Prompt: string; Min, Max : Integer; var Value: Integer): Boolean;
Displays a dialog box with the given Caption and Prompt message. It asks the user to enter an integer value in a text box on the dialog.
The value must be in [Min, Max] range.
If the user presses OK, the entered data is stored in the Value variable and the function returns True.
If the user cancels the dialog, then the return value is False and any entered data is lost.
See also InputQuery.

Event Log Explorer scripting constants

When using Event Log Explorer scripting always use constant names, not values since the values could be changed in the future versions of Event Log Explorer.
E.g. wrong code:

```
  LogView.Sort(2, True);
```

Right code:

```
  LogView.Sort(sfDate, True);
```

Sorting order `constants` (used in TSLogView.Sort)

```
  sfNative   = 0;  // sort by native order (from oldest to newest or
visa versa).
  sfType     = 1;  // sort by type
  sfDate     = 2;  // sort by date and time
  sfTime     = 3;  // sort by time
  sfSource   = 4;  // sort by source
  sfEventID  = 5;  // sort by event id
  sfTask     = 6;  // sort by task category
  sfUser     = 7;  // sort by user name
  sfComputer = 8;  // sort by computer name
```

Export `constants` (used in TSLogView.Export)

```
  Exp_HTML   = 0;  // export to HTML
  Exp_TEXT   = 1;  // export to text file
  Exp_EXCEL  = 2;  // export to Excel
```

Column name `constants` (used in TSLogView.Export)

```
  fldnRecordNumber = 'RecordNumber'; // Record number
  fldnTypeStr = 'TypeStr';           // Type (as text)
  fldnDate = 'Date';                 // Date
  fldnTime = 'Time';                 // Time
  fldnTimeStamp = 'TimeStamp';       // Date and time
  fldnEventID = 'EventID';           // Event ID
  fldnSource = 'Source';             // Source
  fldnCategoryName = 'CategoryName'; // Task category
  fldnUserName  = 'UserName';        // User name
  fldnComputer = 'Computer';         // Computer name
  fldnDescr =  'Descr';              // Event description
  fldnData = 'Data';                 // Event binary data
  fldnCustomField1 = 'CustomField1'; // Custom column 1
  fldnCustomField2 = 'CustomField2'; // Custom column 2
  fldnCustomField3 = 'CustomField3'; // Custom column 3
  ...
  fldnCustomField15 = 'CustomField15'; // Custom column 15
```

Event scope `constants` (used in TSLogView.SaveSnapshot, TSLogView.Export)

```
  esAll = 0;            // All events
  esBookmarked = 1;     // Bookmarked events
  esSelected = 2;       // Selected events
```

```
Event type constants
```

```
  EVT_SUCCESS       = 0;   // Success (Information) event type
  EVT_VERBOSE       = 1;   // Verbose event type
  EVT_INFORMATION   = 2;   // Information event type
  EVT_WARNING       = 3;   // Warning event type
  EVT_ERROR,        = 4;   // Error event type
  EVT_CRITICAL,     = 5;   // Critical error event type
  EVT_AUDIT_SUCCESS = 12;  // Success audit event type
  EVT_AUDIT_FAILURE = 13;  // Failure audit event type
```

Do not confuse these constants with Windows API constants like EVENTLOG_SUCCESS, EVENTLOG_ERROR_TYPE, EVENTLOG_AUDIT_SUCCESS etc. They have different values.

Classes

Event Log Explorer scripting is based on object-oriented programming paradigm. It provides several classes to manage the program and event logs.
Sample code:

```
program Sample;
var
  LogView : TSLogView;
  Filter : TSEventFilter;
begin
// TheApp is an instance (object variable) of TSApp class to manage the
application
//OpenLog creates and returns TSLogView object. This log view displays
the System log.
  LogView := TheApp.OpenLog('', 'System', True);
  Filter := LogView.EventFilter;  // EventFilter property returns
TSEventFilter object
  Filter.EVT_ERROR := True; // Set property of TSEventFilter to display
only Errors
  LogView.EventFilterApply;  //Call EventFilterApply method to apply the
filter
end.
```

TSApp

[TSApp is available in Scripting Console only]
TSApp is the Event Log Explorer application class. When a user starts the program, it automatically creates an internal object TheApp of TSApp class. You cannot create TSApp objects manually.

TSApps properties

Name	Type	Access	Description
ActiveView	TSLogView	Read-only	Returns active log view object
ViewCount	Integer	Read-only	Number of the opened log view in the program
ExeName	String	Read-only	Full path to the Event Log Explorer application. You can also use function GetCmdLineParamStr(0)
Title	String	Read-only	Application title
AppName	String	Read-only	Full application name, e.g. Event Log Explorer Forensic Edition.

TSApps methods

function **OpenLog**(Host : String; LogName : String; Wait : Boolean) : TSLogView;
Opens event log LogName from the host Host. If Wait is true, the function waits until log is loaded. If Wait is false, the function returns immediately.
Return value: new TSLogView object or nil if the function fails.

function **OpenLogFile**(FileName : String; Wait : Boolean) : TSLogView;
Opens event log file FileName. If Wait is true, the function waits until log file is loaded. If Wait is false, the function returns immediately.
Return value: new TSLogView object or nil if the function fails.

function **AddLogToView**(LogView : TSLogView; Host : String; LogName : String; Wait : Boolean) : Boolean;
Appends event log LogName from the host Host to a logview LogView. If Wait is true, the function waits until log is loaded. If Wait is false, the function returns immediately.
Return value: True in case of success or False if the function fails.

function **AddLogFileToView**(LogView: TSLogView; FileName : String; Wait : Boolean) : Boolean;
Appends event log file FileName to a logview LogView. If Wait is true, the function waits until log file is loaded. If Wait is false, the function returns immediately.
Return value: True in case of success or False if the function fails.

function **CreateLogMerger**(Host: String; LogNames: TStringList; Wait: Boolean): TSLogView;
Opens and merges event logs specified in LogNames from host Host. If Wait is true, the function waits until all log are loaded. If Wait is false, the function returns immediately.
Return value: new TSLogView object or nil if the function fails.

function **CreateLogFileMerger**(FileNames : TStringList; Wait: Boolean) : TSLogView;
Opens and merges event log files specified in FileNames. If Wait is true, the function waits until all event log files are loaded. If Wait is false, the function returns immediately.
Return value: new TSLogView object or nil if the function fails.

function **OpenSnapshot**(FileName: String; Wait: Boolean): TSLogView;
[Available in Forensic Edition only]
Opens snapshot specified by FileName. If Wait is true, the function waits until all event log files are loaded. If Wait is false, the function returns immediately.
Return value: new TSLogView object or nil if the function fails.

function **GetView**(Index : Integer) : TSLogView;
Returns opened event log view specified by Index or nil if no view available.

TSLogView

[TSLogView is available in Scripting Console only]
TSLogView represents an Event Log Explorer log view.

TSLogView properties

Name	Type	Access	Description
Event	TSEvent	read-only	Returns an current event in the log view
EventFilter	TSEventFilter	read-only	Returns filter object of this event log view
Caption	String	read-only	Event View caption
IsActive	Boolean	read-only	Indicates if the log view is active
IsEOF	Boolean	read-only	Indicates if the current event reached the end of the events. If the log view is empty, IsEOf = True
IsBOF	Boolean	read-only	Indicates if the current event reached the beginning of the events. If the log view is empty, IsBOf = True
IsCurrentEventBookmarked	Boolean	read-only	Returns true if the current event is bookmarked
IsCurrentEventSFMarked	Boolean	read-only	Returns true if the current event is marked for special filtering
EventCountLoaded	Int64	read-only	Returns the number of loaded events in the log view
EventCountFiltered	Int64	read-only	Returns the number of displayed events in the log view
EventCountBookmarked	Int64	read-only	Returns the number of bookmarked events

TSLogView methods

procedure **Close**;
Closes the log view

procedure **Activate**;
Activates the log view

procedure **GoTop**;
Moves to the first event in the list.

procedure **GoNext**;
Moves to the next event in the list.

procedure **GoPrev**;
Moves to the previous event in the list.

procedure **GoBottom**;
Moves to the last event in the list.

procedure **BookmarkCurrentEvent**;
Bookmarks the current event.

procedure **UnbookmarkCurrentEvent**;
Unbookmarks the current event.

procedure **UnbookmarkAll**;
Unbookmarks all events.

procedure **GoNextBookmark**;
Moves to the next bookmark.

procedure **GoPrevBookmark**;
Moves to the previous bookmark.

procedure **EventFilterApply**;
Applies event filter specified by EventFilter property.

procedure **SpecialFilterMarkCurrent**;
Mark current event for filtering.

procedure **SpecialFilterUnmarkCurrent**;
Unmark current event for filtering.

procedure **SpecialFilterMarkAll**;
Mark all events for special filtering.

procedure **SpecialFilterUnmarkAll**;
Unmark all events for special filtering.

procedure **SpecialFilterApply**;
Apply the special filter - show all specially marked events.

procedure **ClearFilter**;
Clear filter (general and special).

procedure **BeginBulkOperation**;
Signals about the bulk operation - this will detach Event Log Explorer UI from the event list. Use this always when you perform operations with events in a loop.
Example:

```
var
  Count_1001 : Integer;
begin
  with TheApp.ActiveView do begin
    BeginBulkOperation();
    GoTop();
    Count_1001 := 0;
```

```
    while not IsEof do begin
     /// some logic, e.g. calculating the number of event 1001
      if Event.EventId = 1001 then Inc(Count_1001);
      GoNext();
    end;
    EndBulkOperation();
  end;
end.
```

procedure **EndBulkOperation**;
Signals about the end of the bulk operation. You must always use this function if you use BeginBulkOperation.

procedure **GetTimeZone**(var LocalTimeZone : Boolean; var UTCOffset : Integer);
Gets logview timezone. LocalTimeZone will return True, if the log view timezone is a local timezone and false if it is not a local timezone. UTCOffset returns time offset from UTC in minutes. For the eastern hemisphere, it will return positive value. For the western hemisphere - negative value.

procedure **SetTimeZone**(LocalTimeZone : Boolean; UTCOffset : Integer; Wait : Boolean);
Sets a new timezone for the log view. If LocalTimeZone = True, it will set the local timezone and UTCOffset value will be ignored. If LocalTimeZone = False, it will set the timezone defined by UTCOffset.
This function reloads events and if Wait is set to True, it will wait until the log view is reloaded.

procedure **Sort**(SortField : Integer; Asc : Boolean);
Sort event view by the specified SortField. If Asc is True, the events will be sorted ascending, else - descending.
See Sorting order constants at Scripting Constants for more information.

procedure **Export**(Target: Integer; FileName: String; Scope: Integer; Columns: TStringList);
Exports events into another format.
Target defines the output format:
  Exp_HTML - export to HTML;
  Exp_TEXT - export to a text file;
  Exp_EXCEL - export to Excel file.
FileName defines name of the output file.
Scope defines the scope of the export (esAll - all eventgs, esSelected - selected events).
Columns defines the list of columns to export. If Columns is nil or empty, it will export the default column set. If you want to use your own column set, you should create TStringList objects and add column names to this list. See Column name constants at Scripting Constants for more information.

procedure **SaveSnapshot**(FileName: String; Scope: Integer; SnapCC : Boolean; SnapTZ : Boolean; Wait: Boolean);
[Available in Forensic Edition only]
Saves the event list as a snapshot.
FileName defines the name of the snapshot file. Scope defines the scope of the snapshot (esAll - all eventgs, esBookmarked - bookmarked events). SnapCC defines whether to save custom column definitions in the snapshot. Set SnapTZ to true if you want to save current timezone in the

snapshot. If Wait is true, the procedure waits until the log view is saved. If Wait is false, the function returns immediately.

TSEvent

[TSEvent is available in Scripting Console only]
TSEvent gives you access to event fields. You can access the current event from a log view using TSLogView.Event property.

TSEvent properties

Name	Type	Access	Description
RecordNumber	Int64	read-only	Record No (EventRecordID) value
EventType	Integer	read-only	EventType code
TypeStr	String	read-only	Event type as string, e.g. 'Information'
DateTime	TDateTime	read-only	Event timestamp (date and time) with respect to the logview timezone
Date	TDateTime	read-only	Event date with respect to the logview timezone
Time	TDateTime	read-only	Event time with respect to the logview timezone
EventID	DWORD	read-only	Event ID
DateTimeStr	String	read-only	DateTime as string
DateStr	String	read-only	Date as string
TimeStr	String	read-only	Time as string
Source	String	read-only	Event source
Task	Integer	read-only	Task category code
Category	String	read-only	Task category as string
UserName	String	read-only	User name
Computer	String	read-only	Computer name
Description	String	read-only	Event description
EvtXML	String	read-only	XML representation of event

TSEvent methods

function **GetCustomField**(Num: Integer): String;
Returns the custom column value (as string) by its index. If no custom column is available, returns an empty string.

function **GetCustomFieldV**(Num: Integer): Variant;
Returns the custom column value (as variant) by its index. If no custom column is available, returns Null.

function **ExtractFromEvtXml**(XPath : String) : String;
Extracts a value from EvtXML using XPath.
E.g. if you have a security log and you want to access ProcessName under EventData, you can use this XPath
'/Event/EventData/Data[@Name="ProcessName"]'

Sample code:

```
program XPath;
begin
```

```
  DebugOut(TheApp.ActiveView.Event.ExtractFromEvtXml(
           '/Event/EventData/Data[@Name="ProcessName"]'));
end.
```

TSEventFilter

[TSEventFilter is available in Scripting Console only]
TSEventFilter lets you set a filter condition. You can access a current log view filter using TSLogView.EventFilter property, or you can create a new filter using the constructor and then assign it to different log view filters.

TSEventFilter properties

Name	Type	Access	Description
EVT_VERBOSE	Boolean	read/write	Filter Verbose events
EVT_INFORMATION	Boolean	read/write	Filter Information events
EVT_WARNING	Boolean	read/write	Filter Warning events
EVT_ERROR	Boolean	read/write	Filter Error events
EVT_CRITICAL	Boolean	read/write	Filter Critical Error events
EVT_AUDIT_SUCCESS	Boolean	read/write	Filter Audit Success events
EVT_AUDIT_FAILURE	Boolean	read/write	Filter Audit Failure events
Sources	TStringList	read-only	List of event sources to filter. Read-only means that you cannot modify the property itself, but you can modify its contents. I.e. Filter.Source := MySources is incorrect, but Filter.Sources.Add('Microsoft-Windows-Security-Auditing') is correct.
SourcesExclude		read/write	Exclude events with source matched to Sources list
Categories	TStringList	read-only	List of categories to filter
CategoriesExclude	Boolean	read/write	Exclude events with category matched to Categories list
Users	TStringList	read-only	List of users to filter
UsersExclude	Boolean	read/write	Exclude events with username matched to Users list
Computers	TStringList	read-only	List of computers to filter
ComputersExclude	Boolean	read/write	Exclude events with computer matched to Computers list
IDs	String	read/write	List of Event IDs and event ID ranges to filter
IDsExcept	Boolean	read/write	Exclude events with IDs specified by IDs
DescrText	String	read/write	Filter by description text
DescrTextRegExp	Boolean	read/write	Treat DescrText as a regular expression
DescrTextExclude	Boolean	read/write	Exclude events with descriptions matched to DescrText
ByDate	Boolean	read/write	Enable filtering by date
ByTime	Boolean	read/write	Enable filtering by time
DateTimeSeparately	Boolean	read/write	Enable filtering by date and by time separately
FromTime	TDateTime	read/write	Filter from date and time
ToTime	TDateTime	read/write	Filter to date and time
DateTimeExclude	Boolean	read/write	Exclude events matching to specified Date&Time conditions

LastDays	Integer	read/write	Last days filtering
LastHours	Integer	read/write	Last hours filtering
LastHoursExclude	Boolean	read/write	Exclude events matching last days/hours

TSEventFilter methods
constructor Create;
creates a filter object.
You can create a filter object as follows:

```
var
  Filter : TSEventFilter;
begin
  Filter := TSEventFilter.Create;
.....
  Filter.Free;
end.
```

Commonly you don't need to create TSEventFilter object yourself. You should use existing TSLogView.EventFilter object.

procedure **Free**;
Releases TSEventFilter object. You should always call this function when you created TSEventFilter yourself, but you should never call it for existing EventFilter objects.

procedure **Clear**;
Resets the filter to the default (not filtering) value.

procedure **Assign**(Source: TSEventFilter);
Sets all the properties to be equal to the Source properties.

Example:

```
var
  LogView : TSLogView;
begin
  LogView := TheApp.ActiveView;   // Active view
  LogView.EventFilter.Clear;      // Clear current filter if exists
  LogView.EventFilter.IDs := '1000-2000'; // event range
  LogView.EventFilter.EVT_ERROR := True;  // AND only Error events
  LogView.EventFilterApply;       //Apply filter
end.
```

TSXML

[TSXML is available in Scripting Console only]
TSXML is a helper class to query XPath values from XML. You can still use TSEvent.ExtractFromEvtXml function, but if you want to get many XPath queries to the same XML, using TSXML would work faster.

TSXML properties

Name	Type	Access	Description
XML	String	read/write	XML text

TSXML methods

constructor **Create**;
Creates TSXML object. You must always call this constructor before working with TSXML objects.

procedure **Free**;
Destroys TSXML object.

function **XPathQuery**(XPath: String): String;
Returns XPath value.

Example:

```
var
  MyXML : TSXML;
begin
  MyXML := TSXML.Create;
  try
    MyXML.XML := TheApp.ActiveView.Event.EvtXML;
    DebugOut('Keywords : ', MyXML.XPathQuery('/Event/System/Keywords'));
    DebugOut('Level : ', MyXML.XPathQuery('/Event/System/Level'));
  finally
    MyXML.Free;
  end;
end.
```

TStringList

[TStringList is available in Scripting Console and Custom Column formulas]
TStringList is an auxiliary class to work with string list. Note that since TStringList is a standard PascalScript class, it has prefix T, not TS.

You can access a list element using index starting from 0. E.g. SL[0] returns the first element in the list.

TStringList properties

Name	Type	Access	Description
Strings	String, indexed, default	read/write	Accesses to index element of TStringList. Index gives the position of the string, where 0 is the first string, 1 is the second string, and so on.
Sorted	Boolean	read/write	Specifies whether the strings in the list should be automatically sorted.
Duplicates	TDuplicates	read/write	Specifies whether duplicate strings can be added to sorted lists. It can get one of the following values: dupIgnore - Ignore attempts to add duplicate strings to the list. This is default value for Duplicates dupError - raises an exception when an attempt is made to add duplicate strings to the sorted list. dupAccept Permit duplicate strings in the sorted list.
Count	Integer	read-only	Returns the number of elements in the string list.
CommaText	String	read/write	Lists the elements in the string list in a single comma-delimited string.
Text	String	read/write	Lists the strings in the TStrings object as a single string with the individual strings delimited by carriage returns and line feeds.
Values	String, indexed	read/write	When the list includes strings that are name-value pairs, use Values to get or set the value part of a string associated with a specific name part.
Names	String, indexed	read/write	When the list includes strings that are name-value pairs, read Names to access the name part of a string. Names is the name part of the string at Index, where 0 is the first string, 1 is the second string, and so on. If the string is not a name-value pair, Names contains an empty string.

TStringList methods

constructor **Create**;
Creates TStrings object.

procedure **Free**;
Destroys TStrings object.

function **Add**(S: string): Integer;
Adds a new string to the list. Returns the index ()position of the new string in the list.

procedure **Delete**(Index: Integer);
Removes the specified item from the list.

procedure **Clear**;
Removes all items from the list.

function **IndexOf**(const S: string): Integer;
Returns the position of a string (S) in the list.

procedure **Insert**(Index: Integer; S: string);
Inserts a string (S) to the list at the position specified by Index. This operation is not allowed on the sorted list.

procedure **Move**(CurIndex, NewIndex: Integer);
Changes the position of a string in the list. Use Move to move the string at position CurIndex to the position NewIndex.

procedure **LoadFromFile**(FileName: string);
Fills the string list with the lines of text in a specified file.

procedure **SaveToFile**(FileName: string);
Saves the strings of the string list to the specified file (FileName).

Example:

```
var
  SL : TStringList;
begin
  SL := TStringList.Create;
  try
    SL.Add('Application');
    SL.Add('System');
    TheApp.CreateLogMerger('', SL, False);
  finally
    SL.Free;
  end;
end.
```

TSGlobals

[TSGlobals is available in Scripting Console and Custom Column Formulas]
TSGlobals is a global class to save and load your data. When you start the program, it automatically creates an internal object SGlobals of TSGlobals class. You should create TSGlobals objects manually.
Use SGlobals to transfer your data between different scripts.

TSGlobals methods

procedure **SetValue**(Index : Variant; Value : Variant);
Stores Value into the SGlobals storage under index Index

function **GetValue**(Index : Variant) : Variant;
Returns value from the SGlobals storage using index Index.

Example:

```
// Script 1  -  set a Global value
begin
  SGlobals.SetValue('year', 2024);
end.

// Script 2 - check if the year we set in the previous script is a leap
year
begin
  if IsLeapYear( SGlobals.GetValue('year')) then
    DebugOut('Leap year')
  else
    DebugOut('Common year');
end.

// Script 3 - sets filter to Jan 1-Jan 31 of the year we set in Script1
var
  MyXML : TSXML;
begin
  TheApp.ActiveView.EventFilter.ByDate := True;
  TheApp.ActiveView.EventFilter.FromTime :=
EncodeDate(SGlobals.GetValue('year'), 1, 1);
  TheApp.ActiveView.EventFilter.ToTime :=
EncodeDate(SGlobals.GetValue('year'), 1, 31);;
  TheApp.ActiveView.EventFilterApply;
end.
```