The Desaware Event Log Toolkit

Desaware Inc.
1100 E. Hamilton Ave #4
Campbell, CA 95008

www.desaware.com

(408) 377-4770

License Agreement & Warranty

Desaware, Inc.
Software License

Please read this agreement. If you do not agree to the terms of this license, promptly return the product and all accompanying items to the place from which you obtained them.

This software is protected by United States copyright laws and international treaty provisions.

This program will be licensed for use on a single computer. If you wish to transfer the license from one computer to another, you must uninstall it from one computer before installing it on the next. You may (and should) make archival copies of the software for backup purposes.

You may transfer this software and license as long as you include this license, the software and all other materials and retain no copies, and the recipient agrees to the terms of this agreement.

You may not make copies of this software for other people. Companies or schools interested in multiple copy licenses or site licenses should contact Desaware, Inc. directly at (408) 377-4770.

Should your intent be to purchase this product for use in developing a compiled Visual Basic program that you will distribute as an executable (.exe or .dll) file, review the listing of which files (located below and in the File Description section of the product manual) can be distributed and or modified. If Desaware files are included in your executable program, you must include a valid copyright notice on all copies of the program. This can be either your own copyright notice, or "Copyright © 2000 Desaware, Inc. All rights reserved.".

Files: You may distribute event source DLL files created using the Desaware event source utility. You may **not** modify the files listed above in any way.

Source Files: Source code for portions of the Desaware EventLog Toolkit are included for educational purposes only. You may use this source code in your own applications only if they provide primary and significant functionality beyond that included in the toolkit package. You may not use this source code to develop or distribute components and tools that provide functionality similar to all or part of the functionality provided by any of the components or tools included in the Event Log Toolkit package.

Please consult the on-line Help file under the topic File Descriptions for additional information.

Limited Warranty

Desaware, Inc. warrants the physical diskettes or CDs and physical documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the date of purchase.

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskette(s) or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data or use of the software, or special, incidental or consequential damages or other similar claims, even if Desaware, Inc. has been specifically advised of the possibility of such damages. In no event will Desaware, Inc.'s liability for any damages to you or any other person ever exceed the suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

DESAWARE, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, Desaware, Inc. makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty-day duration of the Limited Warranty covering the physical diskettes and documentation only (not the software) and is otherwise expressly and specifically disclaimed.

This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

This License and Limited Warranty shall be construed, interpreted and governed by the laws of the State of California, and any action hereunder shall be brought only in California. If any provision is found void, invalid or unenforceable it will not affect the validity of the balance of this License and Limited Warranty, which shall remain valid and enforceable according to its terms.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Desaware, Inc., 1100 East Hamilton Avenue, Suite 4, Campbell, California 95008.

Introduction

The NT Event log is the preferred way for services and server components to log errors, warnings and other information. Typical uses of the event log include:

- Reporting information in cases where it is not possible or advisable to bring up a message to the current logged on user.
- Reporting information that does not require immediate response.
- Reporting information that you wish to archive for later evaluation or for auditing purposes.

The event log allows you to classify events in several ways, by severity of the error, by source of the error, and by categories that you define. Event viewing tools can then sort or filter event information based on these classifications. The event log system also makes it easy to define events in a manner that is language independent – so events show up correctly in the language of the local system.

The wrong way to use the Event Log with Visual Basic

When Visual Basic added the App.LogEvent method, VB programmers were thrilled to have the ability to easily log events to the event log. The App.LogEvent method could be called easily as shown in this example called from a VB application called "MyVBProject:

```
App.LogEvent "Here is an event logged by VB", vbLogEventTypeError
```

Figure 1 shows the event in the event log entry that results from this call:

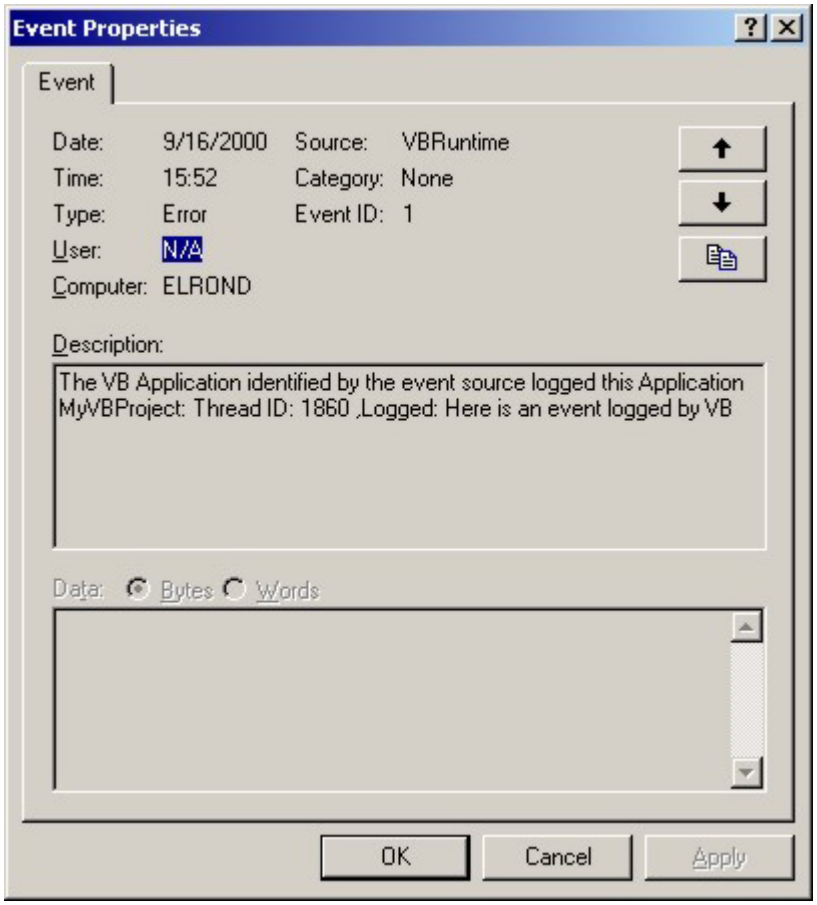


Figure 1 – Typical VB event resulting from a call to the App.LogEvent method.

What's wrong with this picture?

- The Source is VBRuntime – not MyVBProject. So it is impossible to distinguish between events logged by your application and those by any other VB application (See Figure 2).
- The event description is the same for all VB applications. You have to read the description to find the application name and the string used in the LogEvent method.
- The string used in the LogEvent application is language dependent – it will always appear in the event log in the language in which it was written.
- There is no ability to define or report categories, making it impossible to classify event.

In other words, it is impossible to use the event log correctly with the App.LogEvent method.

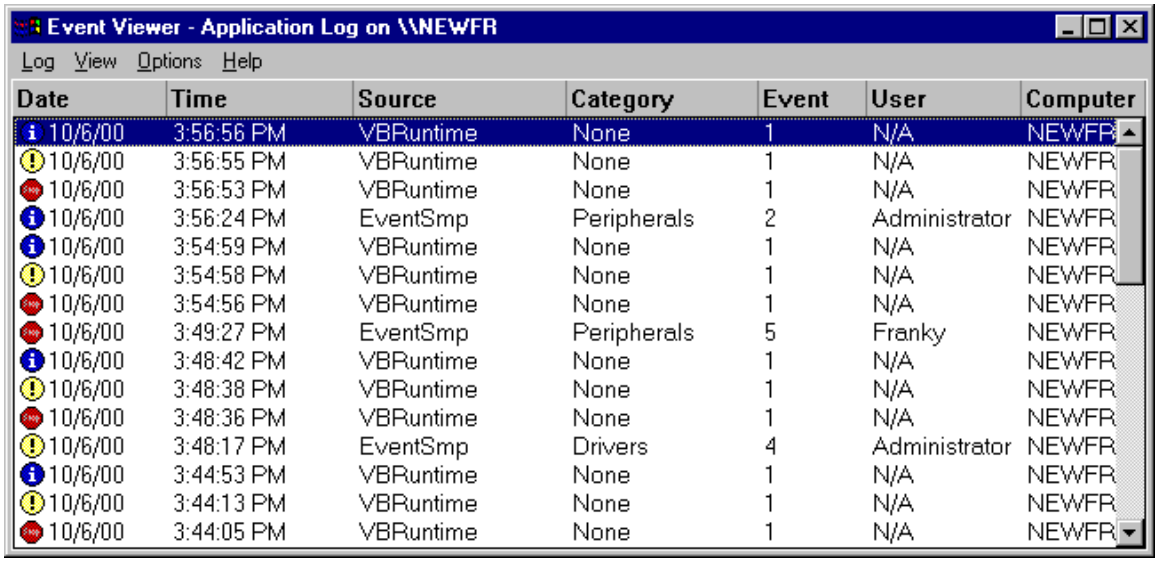


Figure 2 – This image of the Event Log viewer shows that it is impossible to distinguish between events logged by two different VB projects using the App.LogEvent method. Note how VB projects that use event sources (EventSmp) created with this toolkit can be easily identified.

What about Event Log API Functions?

The solution is to use either the event log API functions, or components that wrap those functions. However, as you will soon see, just calling the API functions is not enough. You see, in order to create custom events, you need the ability to create an event source – a special kind of file that contains message and category definitions in the languages that you wish to support. Creating event sources has always been rather complicated and poorly documented. It has also been historically difficult to distribute event sources, as they require specialized registry entries in order to work correctly.

The Right way to use the Event Log from Visual Basic

The Desaware Event Log Toolkit is the first product that not only makes it easy to log events from Visual Basic, it makes it easy to create and distribute custom event sources. In fact, it's so easy to create event sources with our toolkit that even Visual C++ programmers will find it a superior approach to Microsoft's tools.
The Desaware Event Log Toolkit supports the following features:

- Create custom event sources with an interactive Windows application – no complex file formats to learn and edit.
- Event sources support unlimited languages.
- Define custom categories for your event sources.
- Messages are defined by combining identifiers, facilities, and severity information as recommended by Microsoft (you'll read more about this later).
- Automatic generation of VB module and C++ header files with constant definitions.
- Event sources are self-registering – all necessary registry entries can be added automatically using regsvr32.

- Event sources have no component or DLL dependencies – just ship and register.
- Event log class allows access to the event log API – all VB source code included: just drop the class into your project and use it.
- Event log class allows reporting events with all possible parameters – including the difficult to implement ability to specify user accounts with events.

In order to understand how to use the event log, your first step should be to understand exactly what the event log is, and why it works the way it does.

Inside the Windows NT/2000 Event Log

To understand the rather convoluted architecture of the NT/2000 event log, it is necessary to understand the purpose behind its design. First and foremost, the event log was designed to log information produced by system components such as drivers and services. That means that much of the information would have one or more of the following characteristics:

- It would be generated before a user logged on to the system – meaning that notifying a user through a message box was not an alternative.
- It would contain information that was not of immediate interest to a user, but that could be useful to system administrators in evaluating the performance of a system or diagnosing system errors – including errors that did not prevent the system from being used (for example: those that impacted only one subsystem).
- It would contain information that needs to be localized. Since Windows itself is localized to many different languages, and many events are generated by Windows operating system components, localization became an important design factor from the beginning. What's more – it was important to be able to add new languages without modifying each of the system components to support the new language, or having a different version of system components for each language.

To address these issues, Microsoft adopted the event log architecture shown in figure 3.

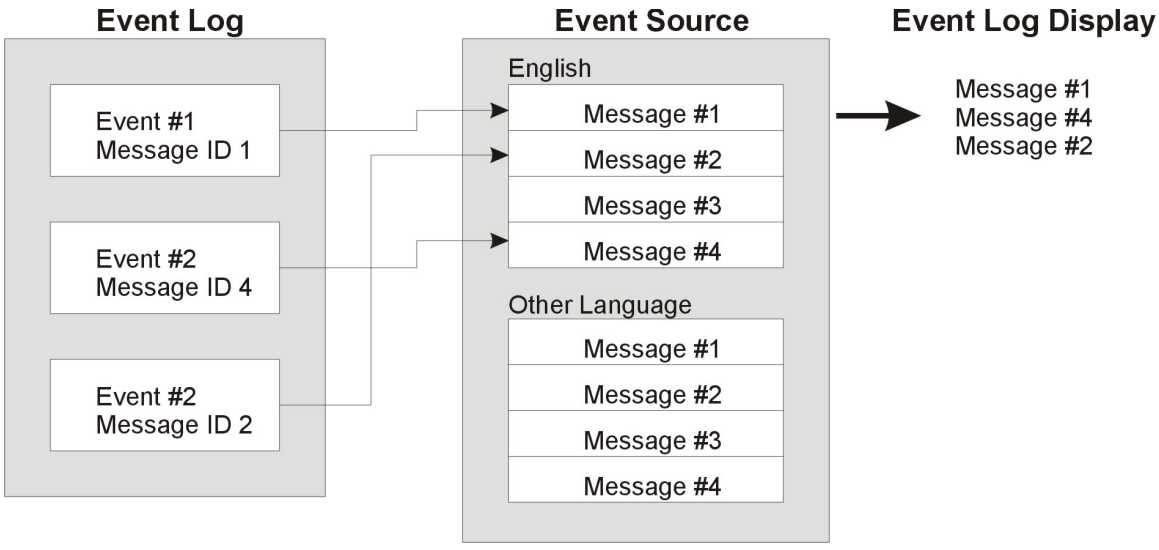


Figure 3 – Event log architecture

The event log itself does not contain any text that needs to be localized. Instead, it contains information identifying an event source, an event identifier, and any language independent text or binary data that it wishes to report. The event source contains the actual message text for every language supported by the event source. Thus, if you support English, French and Japanese, you would find message #1 available in the event source in all three languages.
When the event log is displayed, the event log viewer reads this information from the event log. It opens the event source and looks up the message identifier for the current language being used by the system (the current locale), then displays the correct text. The event log viewer can also merge language independent text in the event log into the messages (you'll read more about this next).

Inside a Message

Because the event log was designed to log events to be viewed at a later time, it was also designed to make it easy to categorize and filter events. And while it is easy for system managers to filter and search for events with the event log viewer, the way those categories are defined from a developer's perspective are less clear (to put it kindly).
When you log an event, the parameters to the ReportEvent method include the following:

- The event source
- The event category
- The severity of the event (error, warning, information, auditing).
- The event identifier
- The user that logged the event (optional)
- Other language independent text
- Binary data.

Let's look at these one at a time.

The Event Source

Each event source has a name and is registered in the system under the registry key

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/EventLog
```

There are three subkeys under this key: *Application*, *Security* and *System*.

The name of the event source appears as a subkey under these three keys, thus if you create an event source named MyEventSource, it would appear in the registry under:

```
HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/EventLog/Applicati
on/MyEventSource
```

An event will appear in the event log under the Application, Security or System event logs depending on where the event source is registered.
The Desaware Event Log Toolkit always registers event sources under the Application key, because the vast majority of people using the toolkit will be creating event sources for applications, components and services – all of whose events should appear in the

Application event log. The System event log is intended for system components and drivers. The Security event log is intended for security audit information.
The MyEventSource registry key contains multiple named values.

Required registry values:

- EventMessageFile – The full path to the event source file.
- TypesSupported – Severity types supported by this event source.

Optional registry values:

- CategoryMessageFile – The full path to the file containing message categories.
- CategoryCount – The number of categories in the file.

The Desaware Event Log Toolkit automatically enters these values in the registry, including the category entries if categories are defined.
The automatic registration built into event log sources created by this toolkit should suit the needs for virtually all situations. However, you always have the option of doing your own event source registration. Thus it is possible, for example, to manually register an event source created with this toolkit for use with the system event log if you are creating a device driver.

The Event Category

An event source may define categories of events. These are always specific to an event source. Categories are numbered from 1 through the number of categories. Each one has a category name (which is also language independent – so each category has a text string for each supported language).

The Event Severity

Events fall into five possible types (or severities):

- Error – Use to indicate a major failure in an application or component.
- Warning – Use to indicate a condition that is not immediately fatal, but that could cause problems if ignored.
- Information – Use to indicate significant events.
- Success Audit – Use to log successful operations that are being audited by the security system.
- Failure Audit – Use to log operations that fail due to security considerations.

Generally speaking, you will only use the first three event types, as security audit information is logged by the operating system depending on the security settings and system policies.

The Event Identifier

Here is where things begin to get tricky.
An event identifier is made up of three different values that are combined using the logical OR operator – the event code, the facility, and the severity. These are laid out in the event identifier as shown here:

```
//  Values are 32 bit values laid out as follows:
//
//   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//  +---+-+-+-----------------------+-------------------------------+
//  |Sev|C|R|     Facility          |               Code            |
//  +---+-+-+-----------------------+-------------------------------+
//
```

The severity value can be one of the following values

- 0 = Success
- 1 = Information
- 2 = Warning
- 3 = Error

The Facility values can be any values you wish to help you to categorize messages. Most will use the default facility code of &HFFF which corresponds to “Application”. Facility codes smaller than 256 are reserved by the system. The “C” and “R” bits are reserved and can be left as zero.

At this point you are probably very confused: If the severity is determined when reporting the event (as described earlier), and the event log (Application, System or Security) is determined by the location of the event source in the registry, what is the relation between the severity and log as reported, and the severity and facility in the event identifier?

There is no relation.

The severity and facility codes in the event identifier are intended to help developers to manage events – especially when there are large numbers of events. Consider the following events that one might specify when creating a service.

ServiceStarted = 1
ServiceStopped = 2
ServiceTCPError = 3
ServiceDNSError = 4
ServiceDied=5
TooManyClients=6

Which of these are serious? Which of them relate to the service itself and which to network connectivity?

Now imagine that the events were reported as follows:

ServiceStarted = SEVERITY_INFORMATION Or Application Or 1
ServiceStopped = SEVERITY_INFORMATION Or Application Or 2
ServiceTCPError = SEVERITY_ERROR Or Network Or 3
ServiceDNSError = SEVERITY_ERROR Or Network Or 4
ServiceDied= SEVERITY_ERROR Or Application Or 5
TooManyClients= SEVERITY_WARNING Or Application Or 6

A programmer reading this code can instantly determine which events are severe, and whether events are associated with the service itself (Application) or a network operation. Obviously, it is in your interest to use the same severity values in the event identifier as you do when reporting the event – doing otherwise could lead to confusion.

The Logging User

You can specify a user account when logging an event. The event logging classes included with the Desaware Event Log toolkit allow you to do so. In most cases users information is not included in the event log (especially services, which typically run under the local system account).

Language Independent Text

The message text in the event source can contain parameters which allow you to merge in additional text provided when reporting the event. Text that you merge in using ReportEvent should be language independent (file names, for example).
The parameter locations are identified by escape sequences that are indicated by the % character. The following escape sequences are supported:

%0 – Ends the text without a newline (crlf) character.
%n – Merges in string #n specified by the ReportEvent function. Thus if you pass three strings to ReportEvent, %2 will merge the second of these strings into the message text at the location of the %2.
%n!printformat! – Like %n, except that the data is formatted according to the printformat characters. These characters are the same as those used by the printf command in the C language (refer to your online documentation or MSDN for a complete list of print format specifiers).
%% - Replaced by the '%' character.
%n - Add line break.
%r – Adds return (without a newline character).
%*space* – Adds a space character.
%. – Adds a period (without terminating the message).
%! – Adds an exclamation point.

Examples:

Message	String Parameters	Result
Operation %1	Succeeds	Operation Succeeds
Operation %1 %n %2 %!	Succeeds, Reboot	Operation Succeeds Reboot!
Function %1 %% Complete	90	Function 90 % complete

Binary Data

You can attach any arbitrary binary data to an event when reporting the event. The binary data will be displayed in hexadecimal by the event viewer.

Using the Desaware Event Source Utility

The Desaware Event Source utility creates an Event Source file based on the event source name, messages, severity, categories, facilities and languages specified. The resulting Event Source file is compiled as a self-registering DLL file that does not require any additional dependency files.

Creating an Event Source file - Quick Start

To create an Event Source file, you need to specify the Application (or Event Source) name, version information for the Event Source, and at least one event message.

Select the Event Source utility's **Edit** – **Version Resources** menu and enter the version resource strings for your compiled Event Source file. At a minimum, you must enter the File Version Number and Company Name. Select the Ok button when finished.

Select the Event Source utility's **Edit** – **Registry Settings** menu and enter the Application (Event Source) name. This will typically be the name of the application using the event source, but can be any name you wish. Select the Ok button when finished.

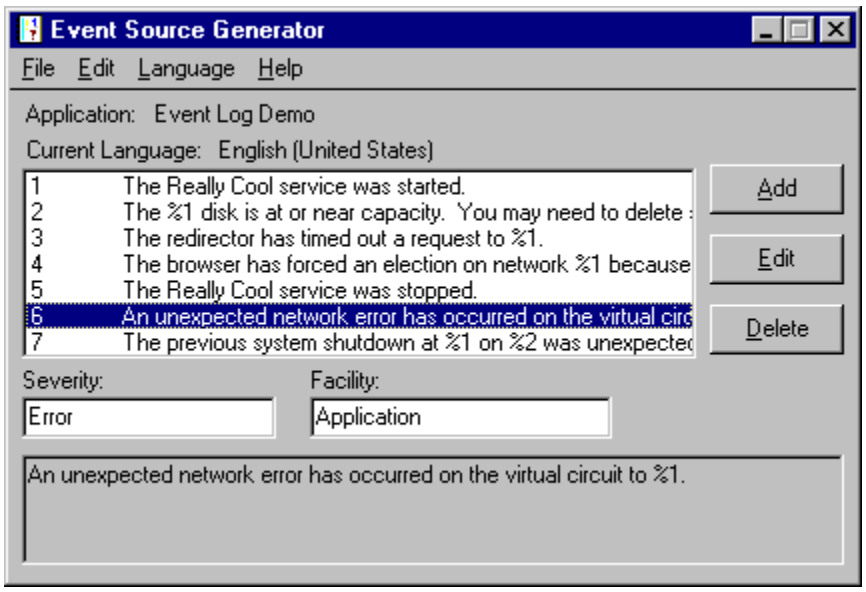
Select the Add button to add an event message. Select a Severity, Facility, and Message ID number for your message. Enter the message string, then select the Ok button when finished. Repeat this step until you have finished adding all the event messages you need.

You can save your Event Source Project file before building the Event Source file by selecting the **File** – **Save As** menu command.

Select the **File** – **Build Source** menu to build the Event Source file. Enter the destination path and file name for the Event Source or select the Browse button to select the destination path and file name. Select the Ok button when finished.

Event Source main Form

The main Event Source utility form displays the current Application name for the Event Source, the currently select Language for the Event Messages, and the current Event Messages. Selecting the Add button allows you to add a new Event Message for the selected language. The Edit button allows you to edit the selected Event Message for the selected Language. The Delete button deletes the selected Event Message for the selected language.



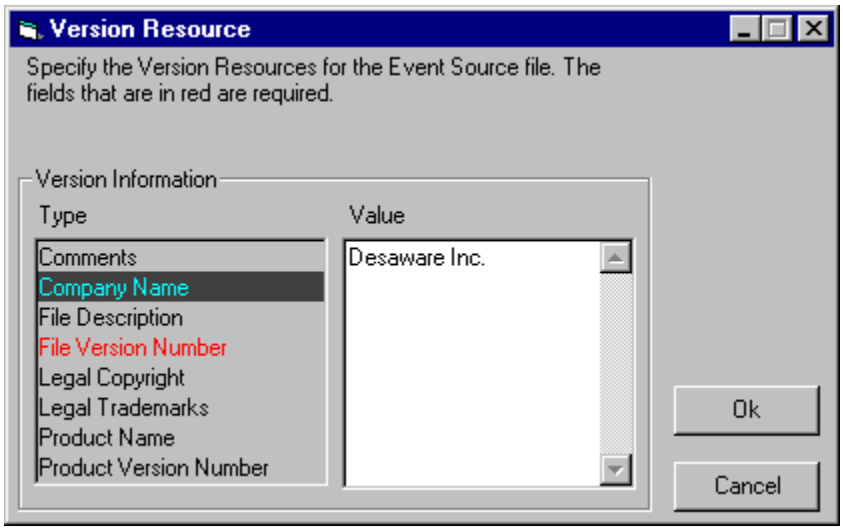
Registry Settings Form

The Registry Settings form holds the information to be written into the registry when the event source file is registered. Enter the Event Source name in the Application text box. Select from Application, System, or Security as the Log type of your Event Source file (you should use the Application log for all applications and services). Check the appropriate Events check boxes to select the types of severity your event source will support. Select the Ok button to save the changes, or the Cancel button to cancel the changes.

Version Resources Form

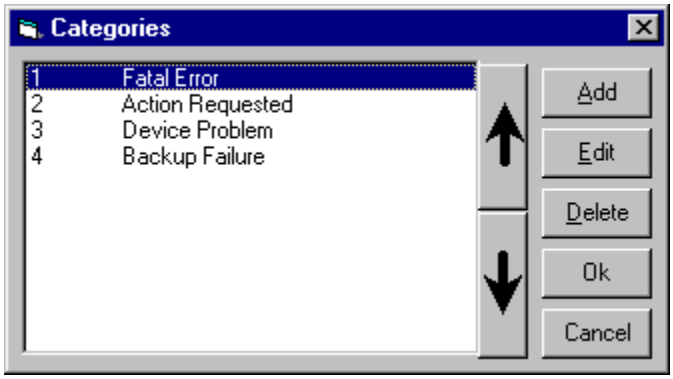
The Version Resources form is used to hold the version resources that are written to the Event Source file. The Company Name and File Version Number fields are required. All of the other version fields are optional but recommended. The File Version Number and Product Version Number fields should be formatted as “#.#.#.#” (for example 1.2.3.4). Select an entry from the Version Resource Type list box, then enter the value for that

field in the Version Resource Value text box. Select the Ok button to save the changes, or the Cancel button to cancel the changes.



Categories Form

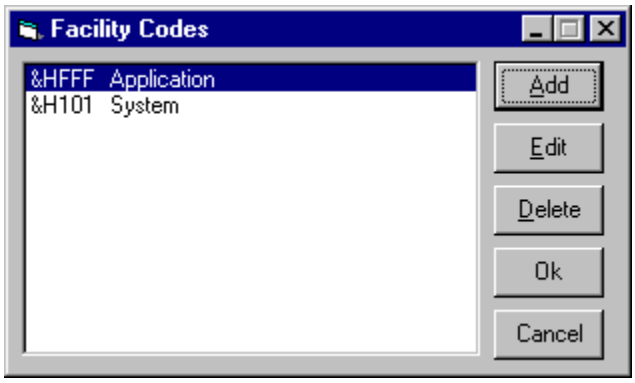
The Categories form allows you to specify custom categories for your Event Source file. The list box displays the existing Categories. The Categories are automatically assigned a Category number when they are added. The Category numbers begin with 1 and increment by 1 for each additional category. Select a Category in the list box and the Up or Down arrow buttons to rearrange the Categories. Select the Add button to add a new Category to the list. The Category name is limited to 20 characters. Select the Edit button to change the name of the selected Category. Select the Delete button to delete the selected Category. Select the Ok button to save the changes, or the Cancel button to cancel the changes.



Facilities Form

The Facilities form allows you to specify custom facilities for your Event Source file. The list box displays the existing Facilities and their values. Select the Add button to add a new Facility to the list. Select the Edit button to change the name or value of the selected Facility. The Facility name is limited to 20 characters. The Facilities values must be between 256 and 4095. The value 4095 is normally used for the Application Facility.

Select the Delete button to delete the selected Facility. Select the Ok button to save the changes, or the Cancel button to cancel the changes.



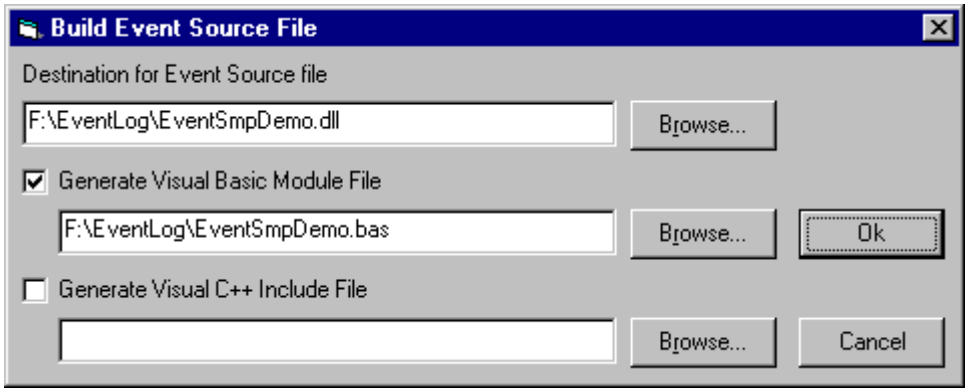
Select Languages form

The Select Languages form allows you to add or remove support for additional languages in your Event Source. Each language includes its own list of Event Messages, Categories, and Facilities. Select the Add button to add a new Language to the list. When you add a new language, a copy of the existing Event Messages, Categories, and Facilities for the current language is made and added to the new language (you should then edit each of these entries, translating them into the new language). Select the Delete button to delete the selected language. When you delete a language, all of the language's Event Messages, Categories, and Facilities are deleted. Select the Ok button to close the form.

Tip: If you will be adding multiple language support, you should complete the Event Messages, Categories, and Facilities for one language first. Then you can add another language which will be created with a copy of the same Messages, Categories, and Facilities numbers as the first language.

Build Event Source form

The Build Event Source form allows you to compile your Event Source file. It also includes options to output the defined Message numbers, Category numbers, and Facility numbers to a Visual Basic Module file or a Visual C++ header file. Specify a destination path and file name for your Event Source file or use the Browse button to select a path and file name. Select the Generate Visual Basic Module File or Generate Visual C++ Include File check boxes to output the Message numbers, Category numbers, and Facility numbers. Similarly, specify a destination path and file name for the Visual Basic or Visual C++ output files or use the Browse button to select a path and file name. Select the Ok button to build the Event Source file and output files. Select the Cancel button to cancel the build.



Event Source Project file

The information you enter using the Event Source utility can be saved into an Event Source Project file. Event Source Project files can be opened, saved, and created using the Event Source utility.

Event Source menu commands

- File
  - New – Creates a new Event Source Project file.
  - Open – Opens an Event Source Project file.
  - Save – Saves the current Event Source information into the Event Source Project file.
  - Save As – Saves the current Event Source information into the specified Event Source Project file.
  - Build Source – Displays the Build Event Source form.
  - Exit – Closes the Event Source utility.
- Edit
  - Version Resource - Displays the Version Resources form.
  - Registry Settings - Displays the Registry Settings form.
  - Categories - Displays the Categories form.
  - Facilities - Displays the Facilities form.
  - Languages - Displays the Languages form.
- Language
  - Contains one or more Languages for the current Event Source. Select a language from this list to select the current working language for the Event Source.
- Help
  - Help Topics – Displays the Desaware Event Log Toolkit help file.
  - About – Displays version information for the Event Source utility.

The Desaware Event Log API Class

The Event Log API functions can be tricky to call from Visual Basic. Therefore the Desaware Event Log Toolkit includes the dwEventLog class which provides wrappers for most of the event log API functions. This class is provided as source code so you can simply add it to any project.
To use the class, simply create a new object of type dwEventLog thus:

Dim el as New dwEventLog

The dwEventLogTypes Enumeration

The dwEventLogTypes public enumerated value is used to specify the severity of an event.

```
Public Enum dwEventLogTypes
   EVENTLOG_ERROR_TYPE = &H1
   EVENTLOG_WARNING_TYPE = &H2
   EVENTLOG_INFORMATION_TYPE = &H4
   EVENTLOG_AUDIT_SUCCESS = &H8
   EVENTLOG_AUDIT_FAILURE = &H10
End Enum
```

ReportEvent

Use this method to log an event into the event log.

```
Public Sub ReportEvent(ByVal Source As String, _
   ByVal SourceMachine As String, ByVal EventType As dwEventLogTypes, _
   ByVal Category As Integer, ByVal EventID As Long, _
   Optional MergeStringsArray As Variant, _
   Optional BinaryDataArray As Variant, Optional UserName As String, _
   Optional UserMachine As String)
```

The parameters are as follows:

- Source – The name of the event source
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.
- EventType – The severity of the event. Select from the dwEventLogTypes enumerator.
- Category – The category number in the source. Zero for no category.
- EventID – The event identifier.
- MergeStringsArray – A zero based variant containing an array of strings (strings are numbered from 1 to N – leave array index zero empty)
- BinaryDataArray – A variant containing a byte array (from 0 to N – position 0 in the array is valid).
- UserName – The name of the user to associate with the event.
- UserMachine – The server on which the user is valid.

GetNumberOfEventLogRecords

Retrieve the number of records in the specified event log.

```
Public Function GetNumberOfEventLogRecords(ByVal Source As String,_
   ByVal SourceMachine As String) As Long
```

The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be read.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.

GetOldestEventLogRecord

Retrieves the record number of the oldest record in the event log.

```
Public Function GetOldestEventLogRecord(ByVal Source As String,
    ByVal SourceMachine As String) As Long
```

The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be read.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.

ReadEventLog

This method reads an entry in the event log into the dwEventLog object.

```
Public Sub ReadEventLog(ByVal Source As String, _
    ByVal SourceMachine As String, ByVal EventIndex As Long)
```

The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be read.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.
- EventIndex – The number of the event in the event log (starting from zero).

Once an event has been loaded into the object using this method, you can retrieve information about the event using the following methods:

EventSource

The event source for this event.

EventComputer
The computer containing the event source for this event.

EventID
The event identifier for the event

EventString
The formatted message for the event (with parameter strings merged in, as it would be displayed by the event log viewer).

EventCategory
The category number for this event

EventCategoryString
The name of the category for this event.

EventType
The severity of the event.

EventBinary
Any binary data associated with this event

EventUser
The user (if any) that recorded this event.

EventDomain
The domain for the user that recorded this event.

EventGenerated
The date and time at which the event occurred.

EventWritten
The date and time at which the event was recorded in the event log.

EventRecordNumber
The absolute record number of this event.

InsertionStringCount
The number of insertion strings found for this event.

InsertionString(ByVal stringindex As Long)
The insertion string as specified by stringindex. stringindex begins at zero.

WasEventSourceFound

True if the Event Source file was found, False otherwise.

Note:

The ReadEventLog function uses the EVENTLOG_SEQUENTIAL_READ flag to move to the specified EventIndex to read. This is due to a Microsoft bug where for large Event Log files (Microsoft mentions 2MB but our testing showed that this fails on smaller Log files), the EVENTLOG_SEEK_READ method fails. We have found that the speed differences between the two methods are insignificant. Also note that this function is not optimized for reading all records in a log file since it opens and closes the log file for each read operation.

BackupEventLog

Backs up an event log into a file.

```
Public Sub BackupEventLog(ByVal Source As String, ByVal SourceMachine
As String, ByVal BackupFileName As String)
```

The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be backed up.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.
- BackupFileName – The full path and name of the file in which to save the event log. This function will fail if the file already exists.

ClearEventLog

Clears the specified event log.

```
Public Sub ClearEventLog(ByVal Source As String, ByVal SourceMachine As
String, ByVal BackupFileName As String)
```

The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be cleared.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.
- BackupFileName – The full path and name of the file in which to save the event log before clearing it.

IsEventLogFull

Determines if the event log is full. This function is valid in Windows 2000 or later only. An error will be raised if this function is called outside of Windows 2000.

```
Public Function IsEventLogFull(ByVal Source As String, ByVal
SourceMachine As String) As Boolean
```

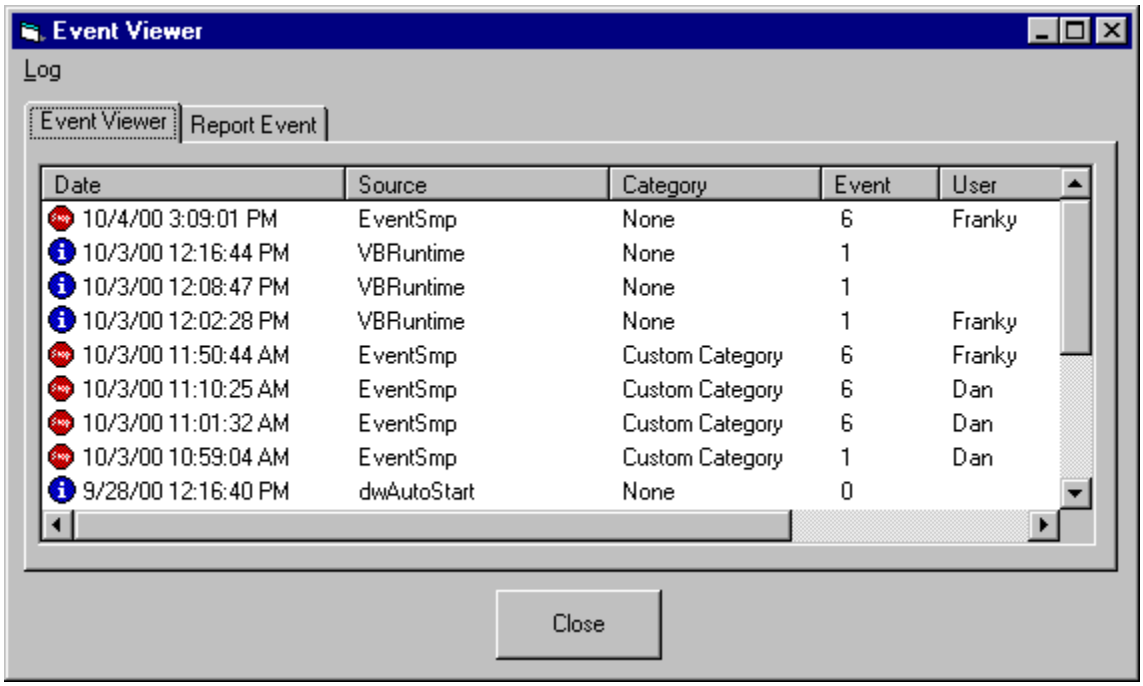
The parameters are as follows:

- Source – The name of the event source whose corresponding event log is to be tested.
- SourceMachine – The name of the server containing the event source. Use an empty string for the current machine.

Using the Desaware Event Viewer and Reporter Utility

The Desaware Event Viewer project demonstrates how to use the Desaware Event Log class. It displays a list of the Events in the Event Viewer tab of the main form. You can double click on an entry to retrieve additional detailed information for each event – such as the event message and additional binary data. The Desaware Event Viewer displays similar information as the Microsoft Event Viewer included with Windows NT.

Note that displaying all the entries for a Log file may take some time depending on how large the Log file is.



Event Viewer

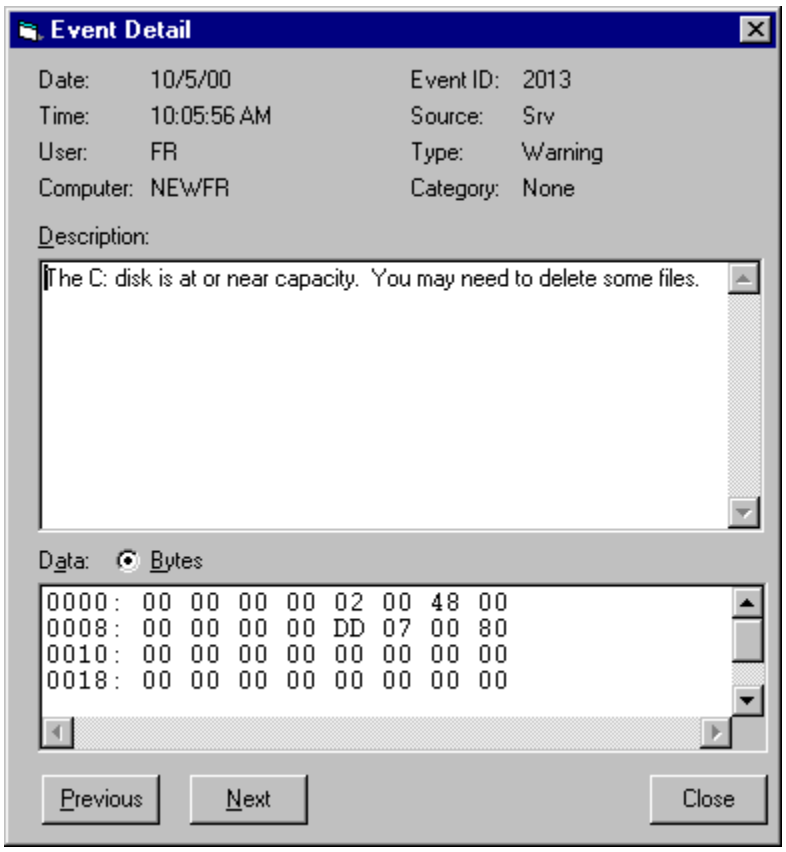
The icons displayed in each event entry identifies the severity of the events as follows:

Information
Error
Warning
Audit Failure
Audit Success

The following columns in the listview control displays the following information:

Date	The date and time that the events were generated.
Source	The name of the Event Source for the events.
Category	The name of the Category for the events.
Event	The event number for the events.
User	The user name for the events, empty if no users were specified with the particular event.
Computer	The computer name for the events.

You can double click on a particular entry to view the message string or binary data for that event. Doing so will display the Event Detail form.



Event Detail Form

The Event Detail form displays the same information as presented in the listview control, with the addition of the message string and binary data for the selected event. Select the next button to view the detail event information for the next event record. Select the Previous button to view the detail event information for the previous event record. The string and binary data are presented as read-only. Select the Close button to exit this form.

Log Menu

The Log menu allows you to specify which type of Event to display. You can choose among the Application, Security, or System events. You may also select to view the events for another computer. The Select Computer menu command displays a list of computers on your network that you may select from. The Refresh menu command causes the Event Viewer to read the event log file again and update the information displayed.

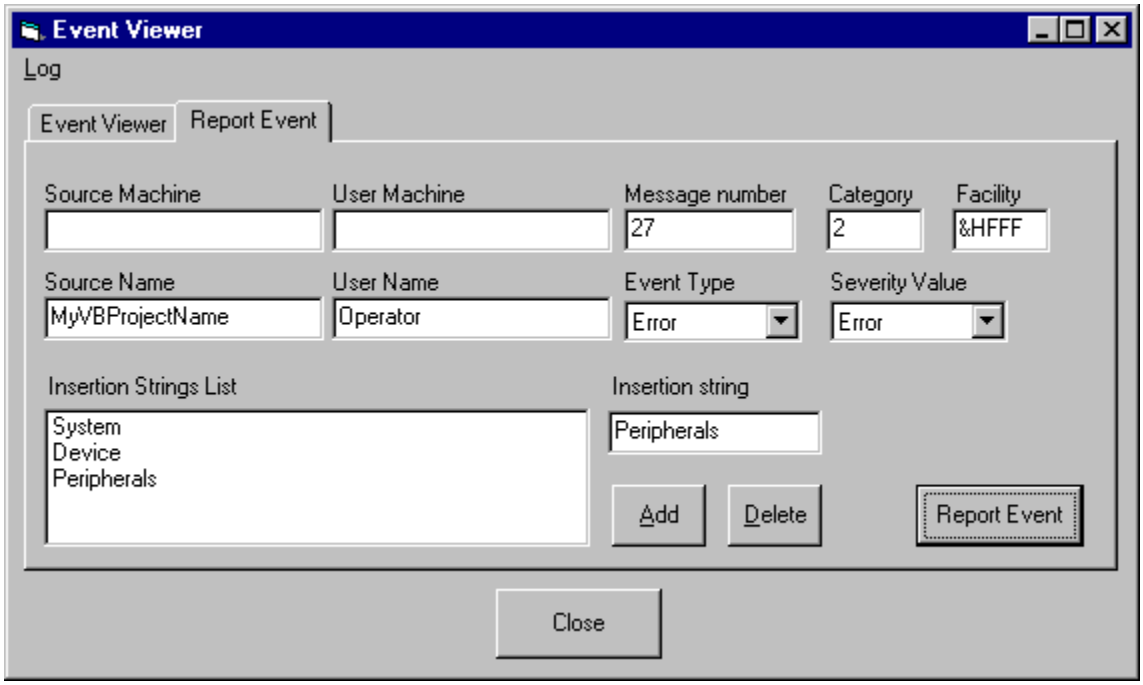
Report Event

The Report Event tab allows you to report an event to the event log. The code demonstrates how to use the Desaware Event Log object's ReportEvent function.

Source Machine – Specify the name of the server containing the event source. Use an empty string for the current machine.
Source Name – Specify the name of the event source.
User Machine – Specify the server on which the user is valid.
User Name - Specify the name of the user to associate with the event.
Message number – Specify the message number in the event source.
Category – Specify the category number in the event source. Zero for no category.
Facility – Specify the facility value in the event source. To specify the facility in hexadecimal, precede the number with "&H".
Event Type – Select the severity of the event.
Severity Value – Select the severity value.
Insertion Strings List – Specify insertion strings for the event source. Insertion strings may be added by entering a string in the Insertion String text box, then selecting the Add button. Insertion strings values must begin with 1 and increment by 1 for each additional insertion string. You may delete an insertion string by selecting it and then selecting the Delete button.

The Facility, Message number, and Severity value are combined together to form the Event ID for the ReportEvent function.

The Report Event button calls the ReportEvent function to log the event.



Redistributable Components

Event sources that you create with this toolkit are redistributable with no royalty fees.

You may incorporate the event log class into your applications and modify it as you wish, however you must include Desaware's copyright notice in the source code everywhere it appears. If you use this source code in a component that you wish to distribute or sell, you must add significant and primary functionality to the component (in other words – under this license you cannot market your own component whose primary task is reporting events into the event log using this source code).

No other files or components of this toolkit may be redistributed.

Technical Support

Desaware prides itself on providing excellent technical support at no charge.
At the same time, while we are glad to address any problems with our software, we know from experience that our software is often used in ways that we never imagined. As enabling technologies (i.e. technologies that allow VB programmers to do things that are beyond the typical VB application), we cannot characterize any of our components or tools for every possible application.

In other words, while we will do our best to address any bugs in our products or issues that look like they have the potential of being bugs, we cannot write your code for you, or debug your program for you. Nor can we provide one on one consulting on particular applications.

When you contact us, we will assume that you are familiar with the material in this manual. We ask that you reduce any problems to the smallest set of code that duplicates the problem.