[MS-EVEN]:
EventLog Remoting Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	2.0	Major	Updated and revised the technical content.
07/03/2007	3.0	Major	Updates for missing content.
07/20/2007	3.0.1	Editorial	Revised and edited the technical content.
08/10/2007	3.1	Minor	Updated the technical content.
09/28/2007	3.2	Minor	Updated the technical content.
10/23/2007	4.0	Major	Updated and revised the technical content.
11/30/2007	4.1	Minor	Updated the technical content.
01/25/2008	5.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

The EventLog Remoting Protocol is a Microsoft-proprietary, **RPC**-based protocol that exposes remote procedure call (RPC) methods for reading **events** in both **live event logs** and **backup event logs** on remote computers. The protocol also specifies how to get general information on a log, such as the number of **records** in the log, the oldest records in the log, and if the log is full. The protocol may also be used for clearing and backing up both types of **event logs**.

Note Early releases of the EventLog Remoting Protocol have never been assigned a version number. However, newer releases of the EventLog Remoting Protocol have version numbers. For example, the version released with Windows Vista is version 6.0.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Localizable
Network Data Representation (NDR)
Opnum
Registry
Remote Procedure Call (RPC)
RPC Protocol Sequence
Unicode
Universal Naming Convention (UNC)
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Backup Event Log: An **event log** that is read-only, and cannot be added to. **Backup event logs** are typically used for archival purposes or for copying to another computer for use by support personnel.

Category String: A **localizable** human-readable string corresponding to the **event category** of a **record**.

Category Message File: A binary resource file defining **category strings** for **event categories**.

Event: A discrete piece of historical information that may be of interest to administrators of a computer system. An example of an **event** would be a particular user logging on to the computer.

Event Category: An application-specific value used for grouping **events**. For example, an application might use one category for all **events** that occur during startup, and use another category for **events** that occur during shutdown. Other applications might use categories to identify the part of the application that raised the **event**.

Event Description String: A **localizable** human-readable string corresponding to the **record**.

Event Log: A collection of **records**, each of which corresponds to an **event**.

Event Source: An application or component that writes to an **event log**.

EventID: An integer indicating the type of **event**. For example, a user logging on to the computer could be one type of **event** while a user logging off would be another type; and these **events** could be indicated by using distinct **EventIDs**.

Event Message File: A binary resource file defining **unexpanded description strings** for an **event source**.

Live Event Log: An **event log** that can be read or added to.

Parameter String: A **localizable** human-readable string inserted into an **event description string** using the string rendering algorithm defined in section 3.2.4.1.5.2.

Parameter Message File: A binary resource file that defines **parameter strings** for an **event source**.

Record: An **event** that is currently represented in an **event log**.

Registry Key or Registry Subkey: A node in the logical tree of the Windows **registry** data store. For more information, see [MSWINREG]. The term subkey specifies that a key has a parent in the logical tree; for example, "A is a subkey of B."

Unexpanded Description String: A **localizable** string containing replaceable insertion patterns that are expanded by using a string-rendering algorithm, defined in section 3.2.4.1.5, to produce an **event description string**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[ISO-8859-1] International Organization for Standardization, "Information Technology -- 8-Bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1", ISO/IEC 8859-1, 1998, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28245&ICS1=35&ICS2=40&ICS3=

Note There is a charge to download the specification.

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", June 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-EERR] Microsoft Corporation, "ExtendedError Remote Data Structure", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol Specification", June 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-RRP] Microsoft Corporation, "Windows Remote Registry Protocol Specification", August 2007.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

[PE-COFF] Microsoft Corporation, "Microsoft Portable Executable and Common Object File Format Specification", May2006, http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, September 2006, http://www.w3.org/TR/REC-xml

1.2.2 Informative References

[MSDN-ANSI] Microsoft Corporation, "Unicode and Character Sets", http://msdn2.microsoft.com/en-us/library/ms776440.aspx

[MSDN-EVENT] Microsoft Corporation, "Event Logging", http://msdn2.microsoft.com/en-us/library/aa363652.aspx

[MSDN-EVENTS] Microsoft Corporation, "Event Schema", http://msdn2.microsoft.com/en-us/library/aa385201.aspx

[MSDN-FMT] Microsoft Corporation, "FormatMessage", http://msdn2.microsoft.com/en-us/library/ms679351.aspx

[MSDN-LCID] Microsoft Corporation, "Locale Identifier Constants and Strings", http://msdn2.microsoft.com/en-us/library/ms776260.aspx

[MSDN-TRANS] Microsoft Corporation, "Translation Between String Types", http://msdn2.microsoft.com/en-us/library/ms776433.aspx

[MSDN-WINDOWSEVENTLOG] Microsoft Corporation, "Windows Event Log", http://msdn2.microsoft.com/en-us/library/aa385780.aspx

[MSWINREG] Microsoft Corporation, "Registry", http://msdn2.microsoft.com/en-us/library/ms724871.aspx

1.3 Protocol Overview (Synopsis)

1.3.1 Background

Event logs allow applications or the operating system to store historical information that may be of interest to administrators. The information is organized as a sequential set of records, which are referred to as events. An example of an event would be a specific user logging on to the computer. Once a record is written, it becomes an event and is treated as a read-only item, and is never updated again.

The events represented in an event log are referred to as records. Records are composed of fields and are numbered uniquely by one of the fields; that is, the first event has its record number set to 1, the second event has its record number set to 2, and so forth. Logs may be configured to be circular. A circular log is one in which the oldest records are overwritten after the log reaches its maximum size.

A computer may have several event logs. One log might be devoted to security events while another might be for general application use.

Applications or components that write to event logs are known as **event sources**. A single event log might contain events from many event sources. However, a particular event source can write to only a single log. That is, a component that writes to multiple event logs is considered for purposes of this specification to be multiple event sources, with one event source per event log.

Event sources write several kinds of events. For example, a user logging on to the computer could be one kind of event, and a user logging off would be another kind of event. When an event source writes an event, it specifies an **EventID** that indicates what specific kind of event is being written. This EventID is reused whenever another event of this same type is written in the future. An event may optionally contain an **event category**, which commonly expresses an application-specific value that is used for grouping events.

An event log can be either a live event log or a backup event log. A live event log is one that is currently in use and thus can be used for both reading and writing. It can be used to create a backup event log, which is a read-only snapshot of a live event log. Backup event logs are often used for archival purposes or for copying a backup event log from one computer to another for use by support personnel.

1.3.2 EventLog Remoting Protocol

The EventLog Remoting Protocol provides a way to access event logs on remote computers.

For both live event logs and backup event logs, the protocol exposes RPC (as specified in [MS-RPCE]) methods for reading events and for getting general information on the log (such as the number of records in the log, the oldest records in the log, and whether the log is full), and therefore can no longer accept additional events.

For live logs only, the protocol also exposes RPC methods for writing events, clearing logs, and creating backup logs.

The protocol does not provide any methods for configuring either event logs or event sources.

The protocol sequencing model is as follows: The client performs an Open operation, issues other requests, and finally performs a Close operation.

For methods used by this protocol, see section 3.1.4.

1.3.3 Localizable Human-Readable Event Descriptions and Other Strings

Windows server implementations are structured in such a way that event logs are language-neutral, and the **localizable** description strings are built from strings loaded from resource files. In this way, different users can view the same event log in their language of choice. The Windows client implementation (Windows Event Viewer) uses a series of algorithms and heuristics to derive localizable event log names, **event description strings**, and event category strings from the event record as expressed by the EVENTLOGRECORD structure.

See section 3.2.4.1.

1.4 Relationship to Other Protocols

The EventLog Remoting Protocol<1> depends on RPC (as specified in [MS-RPCE]) for message transport. When RPC is used by the Eventlog Remote Protocol, RPC uses named Windows named pipes as its transport mechanism, which in turn rely on the Server Message Block (SMB) Protocol, as specified in [MS-SMB].

1.5 Prerequisites/Preconditions

The EventLog Remoting Protocol has the prerequisites, as specified in [MS-RPCE], as being common to protocols depending on RPC.

A prerequisite for the successful use of the methods defined by this protocol is that the caller has appropriate read/write permissions for the resources held on the server, as specified in section 3.1.4.

1.5.1 Server Requirements to Enable Remote Description String Rendering

Requirements for enabling rendering of remote description strings follow:

- The server MUST support the Windows Remote Registry Protocol, as specified in [MS-RRP]. The remote client MUST have read access to the server's remote **registry**.
- The server MUST implement the 'Server' role of the Server Message Block (SMB) Protocol, as specified in [MS-SMB], and share the appropriate "$" shares (for example, "\\server\C$") for reading, so that the remote client can access the message files.

1.6 Applicability Statement

The EventLog Remoting Protocol<2> is used for accessing event logs, which can be used for many different purposes; for example, recording local security events or recording application start/stop events.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following area:

- Protocol Version: The RPC interface for this protocol has its own version number. Each specific version of the protocol requires one specific version of the RPC interface (for more information, see section 2.1.1). This protocol can be extended by adding RPC messages to the interface with **opnums** higher than those defined in this specification. An RPC client determines whether such methods are supported by attempting to invoke the method; if the method is not supported, the RPC run time returns an "opnum out of range" error, as specified in [C706] and [MS-RPCE]. RPC versioning and capacity negotiation in this situation is as specified in [C706] and [MS-RPCE].

1.8 Vendor-Extensible Fields

There are five vendor-extensible fields relevant to the EventLog Remoting Protocol.

1.8.1 Error Values

Any nonzero return value can represent an error. Vendors SHOULD use the values from the NTSTATUS number space, as specified in [MS-EERR].

The EventLog Remoting Protocol uses NTSTATUS values, as specified in [MS-ERREF] section 4. Vendors SHOULD <3> use these values with their indicated meanings. Vendors are free to choose their own values for this field as long as the C bit (0x20000000) is set, indicating that it is a customer code.

1.8.2 Event Log Names

Each event log has a name that is a **Unicode** string. This name MUST be unique across all event logs on the same server. Event log names SHOULD<4> be prefixed with the name of the entity that created the event log to avoid collisions.

1.8.3 Event Source Names

Each event source also has a name that is a Unicode string. This name MUST be unique across all event sources on the same server. An event source name typically identifies the software product to which a given event applies. Event source names SHOULD<5> be prefixed with a unique value (such as the name of the entity that created the event source) to avoid collisions.

1.8.4 EventIDs

EventIDs are integers that are unique on a per-event source basis. The combination of an event source name and an EventID uniquely identifies a specific kind of event.

1.8.5 Event Categories

Event categories are integers that are unique on a per-event source basis. The combination of an event source name and an event category suffices to uniquely identify a class of events. Unlike EventIDs, the use of the category is optional (defaulting to 0). Categories are used to group events into broader classes than can be done with EventIDs. For example, an application might use a category for all events that occur during startup, and use another category for events that occur during shutdown. Management applications might use categories to identify what part of the component raised the event.

1.9 Standards Assignments

The EventLog Remoting Protocol has no standards assignments, only private assignments made by Microsoft using allocation procedures specified in other protocols.

Microsoft has allocated to this protocol an RPC interface **universally unique identifier (UUID)** (using the procedure specified in [C706]) and a named pipe (as specified in [MS-SMB]). The assignments are as follows.

Parameter	Value
RPC interface UUID	{82273FDC-E32A-18C3-3F78-827929DC23EA}

Parameter	Value
Named pipe	\PIPE\eventlog

2 Messages

The following sections specify how EventLog Remoting Protocol messages are encapsulated on the wire and specify common EventLog Remoting Protocol data types.

2.1 Transport

The EventLog Remoting Protocol uses RPC as the primary transport protocol.

Client remote retrieval and expansion of event description, event category, and **parameter strings** are done as specified in [MS-RRP] and [MS-SMB].

2.1.1 Server

The server RPC interface is identified by UUID 82273FDC-E32A-18C3-3F78-827929DC23EA version 0.0, using the RPC **well-known endpoint** \PIPE\eventlog. The server MUST specify RPC over named pipes (that is, ncacn_np) as the **RPC protocol sequence** to the RPC implementation, as specified in [MS-RPCE]. The server MUST specify the Simple and Protected GSS-API Negotiation Mechanism (SPEGNO) (0x9) or NT LAN Manager (NTLM) (0xA), or both, as the RPC Authentication Service (AS) (as specified in [MS-RPCE]). See [MS-RPCE] section 3.3.1.5.2.2 and [C706] section 13.

2.1.2 Client

The client MUST use RPC over named pipes (that is, ncacn_np), as specified in [MS-RPCE], as the RPC protocol sequence to communicate with the server. The client MUST specify either SPEGNO (0x9) or NTLM (0xA) (as specified in [MS-RPCE]) as the Authentication Service (AS).

2.2 Common Data Types

In addition to RPC base types, the sections that follow use the definitions of BOOL, FILETIME, GUID, SID, and ULONG, as specified in [MS-DTYP].

2.2.1 RULONG

The **RULONG** type is used by the **ElfReadELW** and **ElfReadELA** methods to specify the value for the *NumberOfBytesToRead* parameter.

This type is declared as follows:

```
typedef [range(0, MAX_BATCH_BUFF)]
  unsigned long RULONG;
```

2.2.2 RPC_STRING

The **RPC_STRING** structure defines a string of ANSI characters. ANSI strings are specified in section 2.2.12. All integer fields in the **RPC_STRING** structure MUST be in little-endian byte order (that is, least significant byte first).

```
typedef struct {
```

```
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength)] char* Buffer;
} RPC_STRING,
 *PRPC_STRING;
```

Length: The number of bytes (not the number of characters) in the string. This does not include the null terminator.

MaximumLength: If the string is the empty string, this MUST be set to 0. Otherwise, it MUST be the number of bytes in the string, including the null terminator (that is, it MUST be equal to the Length plus 1).

Buffer: Either a pointer to a buffer containing a null-terminated non-empty ANSI string or NULL to indicate an empty string.

2.2.3 RPC_UNICODE_STRING

The **RPC_UNICODE_STRING** structure defines a string of Unicode (as specified in [MS-DTYP]) characters. All integer fields in the **RPC_UNICODE_STRING** structure MUST be in little-endian byte order (that is, least significant byte first).

```
typedef struct {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength/2), length_is(Length / 2)]
    unsigned short* Buffer;
} RPC_UNICODE_STRING,
 *PRPC_UNICODE_STRING;
```

Length: The number of bytes (not the number of characters) in the string. This does not include the null terminator.

MaximumLength: If the string is the empty string, this parameter MUST be set to 0. Otherwise, it MUST be equal to the Length plus 2. This is equivalent to requiring that this value MUST be the number of bytes (not the number of characters) in the string, plus 2 bytes.

Buffer: Either a pointer to a buffer containing a non-empty [UNICODE] string or NULL to indicate an empty string. There MUST NOT be a null terminator within the first Length bytes of Buffer.

2.2.4 EventType

The **EventType**<6> is a 16-bit field that MUST be one of the values in the following list.

Constant/value	Description
EVENTLOG_SUCCESS 0x00000000	An event that describes the successful operation of an application, driver, or service. For example, when a network driver loads successfully, it may be appropriate to log an Information event. It is generally inappropriate for a desktop application to log each time it starts. This is the same meaning conveyed by

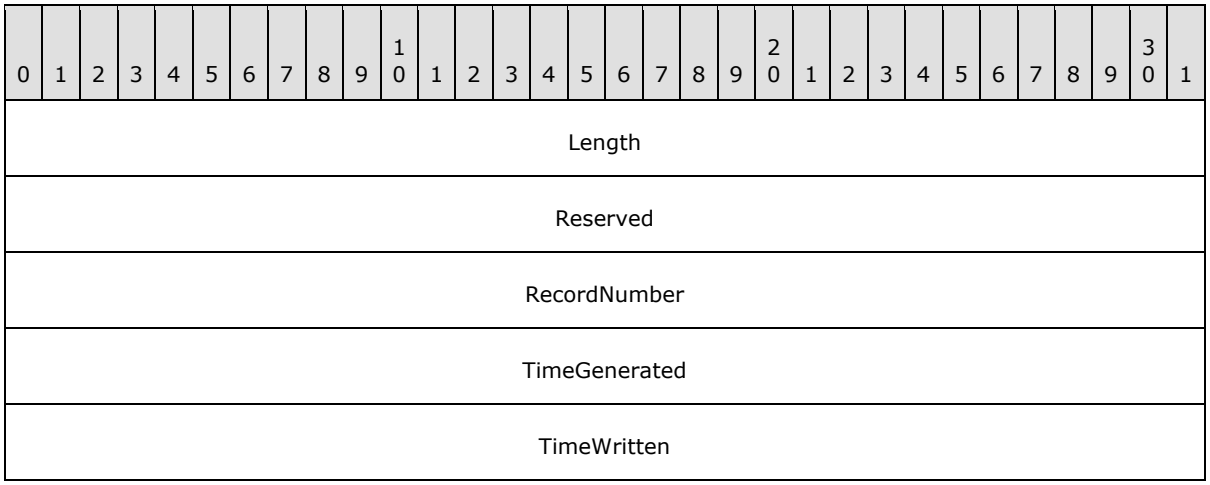
Constant/value	Description
	EVENTLOG_INFORMATION_TYPE.
EVENTLOG_ERROR_TYPE 0x00000001	An event that indicates a problem such as loss of data or loss of functionality. For example, if a service fails to load during startup, an Error event is logged.
EVENTLOG_WARNING_TYPE 0x00000002	An event that is not necessarily significant but may indicate a possible future problem. For example, when disk space is low, a Warning event is logged. If an application can recover from an event without loss of functionality or data, it can generally classify the event as a Warning event.
EVENTLOG_INFORMATION_TYPE 0x00000004	An event that describes the successful operation of an application, driver, or service. For example, when a network driver loads successfully, it may be appropriate to log an Information event. It is generally inappropriate for a desktop application to log each time it starts. This is the same meaning conveyed by EVENTLOG_SUCCESS.
EVENTLOG_AUDIT_SUCCESS 0x00000008	An event that records an audited security access attempt that is successful. For example, a user's successful attempt to log on to the system is logged as a Success Audit event.
EVENTLOG_AUDIT_FAILURE 0x00000010	An event that records an audited security access attempt that fails. For example, if a user tries to access a network drive and is denied access, the attempt is logged as a Failure Audit event.

2.2.5 EVENTLOGRECORD

The EVENTLOGRECORD structure contains information on a single event. This structure is transferred as a set of bytes in the buffer passed in the **ElfrReadELW (section 3.1.4.7)** and **ElfrReadELA (section 3.1.4.8)** methods.

All integer fields in the EVENTLOGRECORD structure MUST be in little-endian byte order (that is, least significant byte first).

The string fields in this structure MUST be ANSI strings when this structure is used with **ElfrReadELA** (section 3.1.4.8) methods, and must be Unicode (as specified in [MS-DTYP]) strings when this structure is used with **ElfrReadELW** (section 3.1.4.7) methods.



EventID	
EventType	NumStrings
EventCategory	ReservedFlags
ClosingRecordNumber	
StringOffset	
UserSidLength	
UserSidOffset	
DataLength	
DataOffset	
SourceName (variable)	
...	
Computername (variable)	
...	
UserSidPadding (variable)	
...	
UserSid (variable)	
...	
Strings (variable)	
...	
Data (variable)	
...	

Padding (variable)
...
Length2

Length (4 bytes): Size in bytes of the structure. The size varies depending on the variable-length fields at the end of the structure.

Reserved (4 bytes): MUST be set to 0x654c664C (which is ASCII for eLfL). This serves as a signature for the structure.

RecordNumber (4 bytes): The record number, as defined in section 1.3.1.

TimeGenerated (4 bytes): Time when the event was generated. The time MUST be expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC). This value is supplied by the event source.

TimeWritten (4 bytes): Time when the event was written. The time MUST be expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC). This value is the time the event was written to the event log.

EventID (4 bytes): EventID generated by the event source, as specified in section 1.8.4.

EventType (2 bytes): Type of the event, as specified in section 1.3.1.

NumStrings (2 bytes): Number of strings in the **Strings** field. This MUST be between 0 and 256, inclusive. A value of 0 indicates that no strings are present.

EventCategory (2 bytes): Event category, as specified in section 1.8.5.

ReservedFlags (2 bytes): Specifies whether or not the last string in the **Strings** field contains well-formed XML, as specified in [XML]. This MUST be one of the two values in this table.

Value	Meaning
0x00000000	The event does not contain XML.
0x00008000	The event contains XML.

ClosingRecordNumber (4 bytes): MUST be set to 0, and ignored on receipt.

StringOffset (4 bytes): This MUST be the offset in bytes from the beginning of the structure to the **Strings** field. If the **Strings** field is not present (NumStrings is 0), this can be any arbitrary value, and MUST be ignored by the client.

UserSidLength (4 bytes): Size in bytes of the user's security identifier, which is located within the **UserSid** field. If there is no **UserSid** field for this event, this field MUST be set to 0.

UserSidOffset (4 bytes): This MUST be the offset in bytes from the beginning of the structure to the **UserSid** field. If the **UserSid** field is not present (that is, if **UserSidLength** is 0), this can be any arbitrary value, and MUST be ignored by the client.

DataLength (4 bytes): This MUST be the size in bytes of the **Data** field. If the **Data** field is not used, this field MUST be set to 0.

DataOffset (4 bytes): This MUST be the offset in bytes from the beginning of the structure to the **Data** field. If the **Data** field is not present (that is, if **DataLength** is 0), this can be any arbitrary value, and MUST be ignored by the client.

SourceName (variable): Variable-length null-terminated string that specifies the name of the source that generated the event, as defined in section 1.8.2. The length of this field is calculated by seeking the NULL that terminates the string.

Computername (variable): Variable-length null-terminated string that assists in identifying the machine that generated the event. This string MUST NOT<7> be interpreted by the protocol, and can be in an arbitrary format.

In practice, the name of the computer. There are no character restrictions on this field's content (for example, a fully-qualified DNS name can be used).

The length of this field is calculated by seeking the NULL that terminates the string.

UserSidPadding (variable): MUST be 0 or more bytes of padding, where the choice of length is implementation dependent. The padding can have any value, and MUST be ignored on receipt.<8>

UserSid (variable): Current user's security identifier, as defined by the **RPC_SID** structure. This parameter can be NULL if the security identifier is not required.

Strings (variable): Zero or more null-terminated strings containing information on the event. The **numStrings** field contains the number of items in this field.

Data (variable): Event-specific binary data. This is supplied by the event source, and MUST NOT be interpreted by the protocol. This data is not always present. The **DataLength** field contains the length of this field. The **DataOffset** field contains the start of this field.

Padding (variable): The **SourceName**, **ComputerName**, **UserSid**, **Strings**, and **Data** fields can all vary in length. The **UserSid**, **Strings**, and **Data** fields MAY be 0 bytes in length. The length of the entire structure up to this point, including these fields, MUST be divisible by 4. Therefore, up to 3 bytes of padding MUST be added to bring the length to a multiple of 4. The padding can have any value, and MUST be ignored on receipt.

Length2 (4 bytes): Same value as the Length field specified as the first member. By having two copies, a buffer containing many events can easily be navigated in both directions.

2.2.6 EVENTLOG_FULL_INFORMATION

The **EVENTLOG_FULL_INFORMATION** structure is used by the **ElfrGetLogInformation (section 3.1.4.18)** method to indicate whether an event log is full or not.

```
typedef struct _EVENTLOG_FULL_INFORMATION {
  unsigned long dwFull;
} EVENTLOG_FULL_INFORMATION;
```

dwFull: If the event log is not full, dwFull MUST be set to 0. If the event log is full, dwFull MUST be set to 1.

2.2.6.1 NT Object Path

A string referred to as an NT Object Path is used by several methods to allow the specification of either a file path that is local to the server or a remote file path.

An NT Object Path string MUST begin with \??\. If the string begins with \??\UNC\, it MUST be interpreted by the server as a **Universal Naming Convention (UNC)** path after replacing the \??\UNC\ with \\. Otherwise, the remainder of the string MUST be interpreted by the server as a local file path in whichever file naming syntax is used by the server's local file system; all characters MUST be considered legal by the EventLog Remoting Protocol, as the string is simply to be passed to the underlying file system.

For example, if the NT Object Path is \??\UNC\wmiscratch\scratch\x.x, the server interprets it as indicating the UNC path \\wmiscratch\scratch\x.x. If the NT Object Path is instead \??\C:\scratch\x.x, the server interprets it as indicating the local file system path C:\scratch\x.x.

2.2.7 RPC_SID

The **RPC_SID** structure is used by methods that write events.

```
typedef struct _RPC_SID {
  unsigned char Revision;
  unsigned char SubAuthorityCount;
  SID_IDENTIFIER_AUTHORITY IdentifierAuthority;
  [size_is(SubAuthorityCount)] unsigned long SubAuthority[*];
} RPC_SID,
 *PRPC_SID;
```

Revision: This member is specified in [MS-DTYP] section **2.2.4.2**.

SubAuthorityCount: This member is specified in [MS-DTYP] section **2.2.4.2**.

IdentifierAuthority: This member is specified in [MS-DTYP] section **2.2.4.2**.

SubAuthority: This member is specified in [MS-DTYP] section **2.2.4.2**.

The fields in this structure exactly correspond to the fields in the SID structure, as specified in [MS-DTYP] section **2.2.4.2**.

2.2.8 Handles

The IELF_HANDLE_type defines a context handle (as specified in [C706]) to the target server.

```
typedef [context_handle] void* IELF_HANDLE;

typedef [context_handle] void** PIELF_HANDLE;
```

2.2.9 EVENTLOG_HANDLE_A and EVENTLOG_HANDLE_W

The event log remote interface on a particular server is referred to by a handle, which can be EVENTLOG_HANDLE_A or EVENTLOG_HANDLE_W. In the specific case of the event log remote interface, the handle is nothing more than the name of the server providing the interface. The name can be specified using either Unicode (as specified in [MS-DTYP]) or ANSI, and the formats are:

```
typedef [handle,unique] wchar_t* EVENTLOG_HANDLE_W;

typedef [handle,unique] char* EVENTLOG_HANDLE_A;
```

Some of the EventLog Remoting Protocol methods (for more information, see section 3.1.4) have an EVENTLOG_HANDLE_W or EVENTLOG_HANDLE_A as their first argument. In these methods, the client maps this string to an RPC binding handle. The server ignores this argument. See [C706] sections 4.3.5 and 5.1.5.2.

2.2.10 RPC_CLIENT_ID

The **RPC_CLIENT_ID** structure is used in the **ElfrChangeNotify (section 3.1.4.21)** method for local method invocations only.

```
typedef struct _RPC_CLIENT_ID {
  unsigned long UniqueProcess;
  unsigned long UniqueThread;
} RPC_CLIENT_ID,
 *PRPC_CLIENT_ID;
```

UniqueProcess: A 32-bit unsigned integer. Ignored when **ElfrChangeNotify** (section 3.1.4.21) is invoked remotely.

UniqueThread: A 32-bit unsigned integer. Ignored when **ElfrChangeNotify** (section 3.1.4.21) is invoked remotely.

2.2.11 Constants Used in Method Definitions

There are several constants that are used in various methods.

Constant/value	Description
MAX_STRINGS 0x00000100	Maximum number of strings a method accepts (typically in a *NumStrings* parameter).
MAX_SINGLE_EVENT 0x0003FFFF	Maximum data size a method accepts for a single event (typically in a *DataSize* parameter).
MAX_BATCH_BUFF 0x0007FFFF	Maximum amount of data in bytes that can be read by a method (typically in a *NumberOfBytesToRead* parameter).

2.2.12 Unicode Versus ANSI String Representations

The EventLog Remoting Protocol supports both Unicode, as specified in [MS-DTYP], and ANSI strings. In this specification, ANSI strings refer to multi-byte strings in which the encoding is controlled by the current system code page<9>.

The server MUST support conversions between character sets. For example, one client might write events using ANSI (multi-byte) strings, and another client might read those same records as Unicode<10>. For how clients choose a character set, see section 3.1.4. The way in which a Unicode string is converted to or from an ANSI string is implementation specific. In the occasional case in which the server cannot convert from Unicode to ANSI, the operation MUST fail.

3 Protocol Details

The EventLog Remoting Protocol asks the RPC runtimes at both the client and server to perform a strict **Network Data Representation (NDR)** consistency check.<11>

3.1 Server Details

The event log server handles client requests for any of the methods, as specified in section 3.1.4, and operates on the logs and the configuration on the server.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to explain how the protocol behaves. This specification does not mandate that implementations adhere to this model as long as their external behavior is consistent with what is described in this specification.

3.1.1.1 Event Log Records

An event log record is the structure that represents an occurrence of an event in the system.

The event log record includes time, type, and category information, and corresponds to the EVENTLOGRECORD (section 2.2.5) structure.

3.1.1.2 Event Logs

The log is a persistent store of event log records. Event logs are of two types: live event logs, which can be written to and read from, and backup event logs, which can only be read from. The backup logs are created using the methods that back up (or copy) a live log to a backup log.

The logs are registered by creating registry entries. For how to create these entries, see [MS-RRP]. Each **subkey** under HKEY_LOCAL_MACHINE\system\services\eventlog results in an event log. The name of the log is the same as the subkey.<12>

The client MUST NOT modify event log registry entries. The server MUST configure those event log registry entries.

3.1.1.3 Event Sources

The event source is intended to identify the software that reports the event. In the EventLog Remoting Protocol, the event source is specified as a string. The server maintains an association between event sources and logs.<13>When a client calls a method such as **ElfrRegisterEventSourceW** to get a handle for writing, the server uses that association to determine what log will receive any events subsequently published using that handle.

By definition, an event source is only associated with one log in the system.

Event sources are registered by creating subkeys in the registry. These subkeys are located under the keys used to define event logs (see section 3.1.1.2). The name of the subkey is the name of the event source. For example, a log named Log1 would be defined by this key:

```
HKEY_LOCAL_MACHINE\
     system\currentcontrolset\services\eventlog\Log1
```

If there were two sources for that log named source1 and source2, there would be the following two keys:

```
HKEY_LOCAL_MACHINE\
     system\currentcontrolset\services\eventlog\Log1\source1
HKEY_LOCAL_MACHINE\
     system\currentcontrolset\services\eventlog\Log1\source2
```

In addition, the name of the event source needs to be added to the REG_MULTI_SZ "Sources" value defined for the event log.

3.1.1.4 EventID

The EventID identifies the specific kind of event; this classification is relative to the event source that logs it<14>.

3.1.1.5 Context Handles

Clients obtain context handles for both reading and writing purposes. The methods for doing so are specified in section 3.1.4. The server MUST maintain a relationship between each particular handle and a particular log. For handles used for writing, the server MUST also maintain the name of the event source so that it can be injected into any events written using the handle. For handles used for reading, the server MUST maintain the position of the last read so that subsequent sequential mode reads can succeed.

3.1.2 Timers

None.

3.1.3 Initialization

At initialization time, the EventLog Remoting Protocol server MUST register the RPC interface and begin listening on the RPC well-known endpoint that is specified in section 2.1. The server then MUST wait for client requests.<15>

3.1.4 Message Processing Events and Sequencing Rules

This section is an overview of the 21 RPC methods used by the EventLog Remoting Protocol. With one exception, there are two versions of each method that have one or more strings in the argument list: One version takes Unicode strings as arguments (such methods are denoted by a 'W' at the end of the method, which is short for Wide), and one version takes ANSI strings as arguments (such methods are denoted by an 'A' at the end of the method, which is short for ANSI). ANSI strings are converted to Unicode strings at the server (as specified in section 2.2.12) before being further interpreted at the server.

The names and opnums of each method are given below as well as a simple description of the method.

Methods in RPC Opnum Order

Method	Description
ElfrClearELFW	Clears event logs.

Method	Description
	Opnum: 0
ElfrBackupELFW	Creates a backup of a live event log. Opnum: 1
ElfrCloseEL	Used to close context handles obtained by the ElfrOpenELW method, ElfrOpenELA method, ElfrOpenBELW method, or ElfrOpenBELA method. Opnum: 2
ElfrDeregisterEventSource	Used to close context handles obtained by the ElfrRegisterEventSourceW method or the ElfrRegisterEventSourceA method. Opnum: 3
ElfrNumberOfRecords	Obtains the number of records in an event log. Opnum: 4
ElfrOldestRecord	Obtains the record number of the oldest record in an event log. Opnum: 5
ElfrChangeNotify	Reserved for local use. Notifies local processes about changes to the event log. Opnum: 6
ElfrOpenELW	Opens a handle to a live event log that can be used for reading or clearing. Opnum: 7
ElfrRegisterEventSourceW	Opens a handle to a live event log that can be used for writing. Opnum: 8
ElfrOpenBELW	Opens a handle to a previously backed up event log. The handle may be used for reading. Opnum: 9
ElfrReadELW	Reads one or more events from an event log. Opnum: 10
ElfrReportEventW	Writes an event to an event log. Opnum: 11
ElfrClearELFA	Clears an event log. Opnum: 12
ElfrBackupELFA	Creates a backup of a live event log. Opnum: 13
ElfrOpenELA	Opens a handle to a live event log that can be used for reading or clearing. Opnum: 14
ElfrRegisterEventSourceA	Opens a handle to a live event log that can be used for writing.

Method	Description
	Opnum: 15
ElfrOpenBELA	Opens a handle to a previously backed up event log that can be used for reading. Opnum: 16
ElfrReadELA	Reads one or more events from an event log. Opnum: 17
ElfrReportEventA	Writes an event to an event log. Opnum: 18
Opnum19NotUsedOnWire	Reserved for local use. Opnum: 19
Opnum20NotUsedOnWire	Reserved for local use. Opnum: 20
Opnum21NotUsedOnWire	Reserved for local use. Opnum: 21
ElfrGetLogInformation	Gets information on an event log. Opnum: 22
Opnum23NotUsedOnWire	Reserved for local use. Opnum: 23
ElfrReportEventAndSourceW	Writes a single event to an event log. Opnum: 24

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined because it does not affect interoperability.

The first group of conceptual operations relates to initiating interaction with an event log, which may be either a backup event log or a live event log (for the distinction between live and backup, see section 1.3). Interaction can be initiated with a live event log for either writing to the event log or for reading or clearing the event log. Because each of the three conceptual operations can use either Unicode or ANSI strings, this accounts for six of the 20 methods.

The second group of conceptual operations relates to interacting with a log by reading from the log, clearing the log, creating a backup of the log, writing to the log, or writing to the log and specifying the name of the source at the time of the write. Four of these five conceptual operations can use either Unicode or ANSI strings as arguments, accounting for another (2x4+1) = 9 of the 20 methods.

The third group of conceptual operations relates to getting metadata on the log: the number of the oldest record in the log, the total number of records in the log, or other information on the log. These account for another three of the 20 methods.

The fourth group of conceptual operations relates to freeing resources maintained on the server to support its interaction with this client. These account for the remaining two of the 20 methods.

The methods are presented in the table above in the same order as this conceptual grouping.

All methods MUST NOT throw exceptions. All return values use the NTSTATUS numbering space; and, in particular, a value of 0x00000000 indicates success, and any other return value indicates an error. All error values MUST<16> be treated the same, unless specified otherwise.

Because the server makes access control decisions as part of the response to Eventlog Remote Protocol requests, the client MUST authenticate to the server, as specified in section 2.1.1. This is the responsibility of a lower-layer protocol, RPC with named pipes (as specified in [C706]); and the access control decisions affecting the Eventlog Remote Protocol are made based on the identity conveyed by this lower-layer protocol.

The RPC interface for the Eventlog Remote Protocol only uses handles of type IELF_HANDLE. There are two groups of functions that can be used to obtain one of these handles. This protocol asks the RPC runtime via the strict_context_handle attribute to reject use of context handles created by a method of a different RPC interface than this one, as specified in [MS-RPCE] section 3.

There are specific methods used for opening handles for reading. These methods are log-oriented. The caller specifies the particular log (such as Application) or the name of a previously backed up log. These methods MUST succeed if the caller has read access, independent of if the caller has write or clear access (security permissions that allow the user to write to or clear the event log). <17>A caller with read access can read events, get log information (such as the number of records or oldest record), and determine if the log is full. These methods are:

- **ElfrOpenELW** (section 3.1.4.3)
- **ElfrOpenELA** (section 3.1.4.4)
- **ElfrOpenBELW** (section 3.1.4.1)
- **ElfrOpenBELA** (section 3.1.4.2)

The **ElfrOpenELA** (section 3.1.4.4) and **ElfrOpenELW** (section 3.1.4.3) methods are used to obtain handles for backing up and clearing event logs.

For writing purposes, a second group is used. In addition to requiring that the caller have Write permission, the methods use the name of the event source to determine the event log to write to. These methods are:

- **ElfrRegisterEventSourceW** (section 3.1.4.5)
- **ElfrRegisterEventSourceA** (section 3.1.4.6)

The 'A' or 'W' suffix in the method name signifies whether the string arguments to the method contain ANSI or Unicode characters. This MUST NOT affect calls to subsequent methods. For example, a handle obtained by using the **ElfrOpenELW** (section 3.1.4.3) method MUST be usable with either **ElfrReadELW** (section 3.1.4.7) or **ElfrReadELA** (section 3.1.4.8).

When opening the handles, the server MUST check for additional rights. For example, the **ElfrRegisterEventSourceW** (section 3.1.4.5) method MUST succeed if and only if the caller has write access, independent of if the caller has read or clear access. However, the handle returned by the server MUST also be associated with the read and clear accesses if they are possessed by the client. Therefore, a handle returned by the **ElfrRegisterEventSourceW** (section 3.1.4.5) method MUST be usable for purposes other than writing if the caller has the appropriate permissions. Similarly, a handle returned via **ElfrOpenELW** (section 3.1.4.3) or **ElfrOpenELA** (section 3.1.4.4) MUST be usable for writing if the caller has write access.

Later in this section, the requirements on the internal state at the server for these methods to succeed are specified as well as the updates to server state caused by each method if the method succeeds.

3.1.4.1 ElfrOpenBELW (Opnum 9)

The **ElfrOpenBELW (Opnum 9)** method instructs the server to return a handle to a backup event log. The caller MUST have permission to read the file containing the backup event log for this to succeed.

```
NTSTATUS ElfrOpenBELW(
  [in] EVENTLOG_HANDLE_W UNCServerName,
  [in] PRPC_UNICODE_STRING BackupFileName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to a Unicode (as specified in [MS-DTYP]) string specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument. See [C706] sections 4.3.5 and 5.1.5.2.

BackupFileName: NT Object Path of the file where the backup event log is located, as specified in section 2.2.6.1.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706] Context Handles. This handle MUST be closed using the **ElfrCloseEL (Opnum 2) (section 3.1.4.19)** method once the handle is no longer needed.

Return Values: The method MUST return STATUS_SUCCESS on success; otherwise, it MUST return a non-zero value.

Note The value of STATUS_SUCCESS is 0x00000000.

In response to this request from the client, the server MUST fail the method if the *BackupFileName* parameter is NULL or empty, or is not a legal NT Object Path.

The server MUST verify that the caller has read access to the file, and MUST fail the method if the caller does not have read access.

The server MUST attempt to open the file, and MUST fail the method if the open does not succeed. The server MUST fail the method if the file exists but does not contain a backed up event log.

If the above checks all succeed, the server MUST attempt to create a handle to the wanted backup event log, and add the handle to its internal table. If successful, the server MUST return the handle via the *LogHandle* parameter.

The server MUST return a value indicating success or failure for this operation<18>.

3.1.4.2 ElfrOpenBELA (Opnum 16)

The **ElfrOpenBELA (Opnum 16)** method instructs the server to return a handle to a backup event log. The caller MUST have permission to read the file containing the backup event log for this to succeed.

```
NTSTATUS ElfrOpenBELA(
  [in] EVENTLOG_HANDLE_A UNCServerName,
  [in] PRPC_STRING FileName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to an ANSI string (see [MSDN-ANSI]) specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

FileName: NT Object Path of the file where the backup event log is located, as specified in section 2.2.6.1.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706], Context Handles. This handle MUST be closed by using the **ElfrCloseEL (section 3.1.4.19)** method once the handle is no longer needed.

Return Values: The method returns STATUS_SUCCESS (0x00000000) on success; otherwise, it returns a non-zero error code defined in Ntstatus.h.

The server MUST return a value indicating success or failure for this operation<19>.

This is identical to the **ElfrOpenBELW (section 3.1.4.1)** method except that the BackupFileName is an ANSI string in this case.

3.1.4.3 ElfrOpenELW (Opnum 7)

The **ElfrOpenELW** method instructs the server to return a handle to a live event log. The caller MUST<20> have permission to read the file that contains the event log for this to succeed.

```
NTSTATUS ElfrOpenELW(
  [in] EVENTLOG_HANDLE_W UNCServerName,
  [in] PRPC_UNICODE_STRING ModuleName,
  [in] PRPC_UNICODE_STRING RegModuleName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to a Unicode (as specified in [MS-DTYP]) string specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

ModuleName: Specifies the event log name, as defined in section 1.8.2, for which a handle is needed.

RegModuleName: This parameter MUST be ignored by the server. Clients MUST specify an empty string.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706] Context Handles.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

In response to this request from the client, the server MUST determine what event log the client is requesting the handle for. The server MUST treat the *ModuleName* parameter as the event log name itself, as specified in section 1.8.2. If the *ModuleName* parameter does not specify a known event log, the server MUST default to requesting access to the application log that MUST always exist. (For more information on the known event log names ("Event Sources"), see section 3.1.1.3).

The server MUST verify that the caller has read access to the event log, and the server MUST fail the operation if the caller does not have read access to the log.

If the checks above are successful, the server MUST attempt to create a handle to the wanted log and add the handle to its internal table. If successful, the server MUST return the handle via the *LogHandle* parameter.

The server MUST return a value indicating success or failure for this operation.

3.1.4.4 ElfrOpenELA (Opnum 14)

The **ElfrOpenELA (Opnum 14)** method instructs the server to return a handle to a live event log. For this to succeed, the caller MUST have permission to read the file that contains the event log.

```
NTSTATUS ElfrOpenELA(
  [in] EVENTLOG_HANDLE_A UNCServerName,
  [in] PRPC_STRING ModuleName,
  [in] PRPC_STRING RegModuleName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to an ANSI string (see [MSDN-ANSI]) specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

ModuleName: Specifies the event log for which a handle is needed.

RegModuleName: This parameter MUST be ignored by the server. Clients MUST specify an empty string.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706] Context Handles.RPC context handles are also specified in the RPC Application Context Handles topic of the protocol specification.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

This is identical to the **ElfrOpenELW (section 3.1.4.3)** method except that the *ModuleName* is an ANSI string in this case.

3.1.4.5 ElfrRegisterEventSourceW (Opnum 8)

The **ElfrRegisterEventSourceW (Opnum 8)** method instructs the server to return a handle to an event log for writing. The caller MUST have permission to write to the file containing the event log for this to succeed. The module name argument specifies the event source that is used to determine the relevant event log as specified below.

```
NTSTATUS ElfrRegisterEventSourceW(
  [in] EVENTLOG_HANDLE_W UNCServerName,
  [in] PRPC_UNICODE_STRING ModuleName,
  [in] PRPC_UNICODE_STRING RegModuleName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to a Unicode (as specified in [MS-DTYP]) string specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

ModuleName: Specifies the event source for which a handle is needed.

RegModuleName: This parameter MUST be ignored by the server. Clients MUST specify an empty string.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706] Context Handles.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

In response to this request from the client, the server MUST determine what event log the client is requesting the handle for. The server MUST treat the ModuleName parameter as the event source name, as specified in section 1.8.3. If the ModuleName parameter does not specify a known event source, the server MUST default to requesting access to the application log that MUST always exist.

Then the server MUST verify that the caller has write access to the event log, and the server MUST fail the operation if the caller does not have write access to the log.

If the checks above are successful, the server MUST attempt to create a handle to the wanted log and add the handle to its internal table. If successful, the server MUST return the handle via the *LogHandle* parameter.

The server MUST return a value indicating success or failure for this operation.

3.1.4.6 ElfrRegisterEventSourceA (Opnum 15)

The **ElfrRegisterEventSourceA (Opnum 15)** method instructs the server to return a handle to an event log for writing. The caller MUST have permission to write to the file containing the event log for this to succeed. The module name argument specifies the event source, which is used to determine the relevant event log, as specified in the following sections.

```
NTSTATUS ElfrRegisterEventSourceA(
  [in] EVENTLOG_HANDLE_A UNCServerName,
  [in] PRPC_STRING ModuleName,
  [in] PRPC_STRING RegModuleName,
  [in] unsigned long MajorVersion,
  [in] unsigned long MinorVersion,
  [out] IELF_HANDLE* LogHandle
);
```

UNCServerName: Pointer to an ANSI string (see [MSDN-ANSI]) specifying the server, as specified in section 2.2.9. The client MUST map this string to an RPC binding handle, and the server MUST ignore this argument, as specified in [C706] sections 4.3.5 and 5.1.5.2.

ModuleName: Specifies the event source for which a handle is needed.

RegModuleName: This parameter MUST be ignored by the server. Clients MUST specify an empty string.

MajorVersion: Major version of the client. This value MUST be set to 1.

MinorVersion: Minor version of the client. This value MUST be set to 1.

LogHandle: Pointer to an event log handle. This parameter is an RPC context handle, as specified in [C706] Context Handles.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

This is identical to the **ElfrRegisterEventSourceW (section 3.1.4.5)** method except that the *ModuleName* parameter is an ANSI string in this case.

3.1.4.7 ElfrReadELW (Opnum 10)

The **ElfrReadELW (Opnum 10)** method reads events from the event log; the server transmits these events to the client and advances the reader's position within the event log associated with this handle. The strings in the returned event MUST be in [UNICODE].

```
NTSTATUS ElfrReadELW(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long ReadFlags,
  [in] unsigned long RecordOffset,
  [in] RULONG NumberOfBytesToRead,
```

```
  [out, size_is(NumberOfBytesToRead)]
    unsigned char* Buffer,
  [out] unsigned long* NumberOfBytesRead,
  [out] unsigned long* MinNumberOfBytesNeeded
);
```

LogHandle: Handle to an event log to read. This parameter is an RPC context handle, as specified in [C706] Context Handles.

ReadFlags: The caller MUST specify whether the read is to start at a specific record or is to proceed from the last record read. The value MUST include one and only one of the following flags.

Value	Meaning
EVENTLOG_SEQUENTIAL_READ 0x00000001	Read operation proceeds sequentially from the last call to the **ElfrReadELW** (section 3.1.4.7) method or the **ElfrReadELA (section 3.1.4.8)** method, using this handle. This flag MUST NOT be used with EVENTLOG_SEEK_READ.
EVENTLOG_SEEK_READ 0x00000002	Read operation proceeds from the record specified by the *RecordOffset* parameter. This flag MUST NOT be used with EVENTLOG_SEQUENTIAL_READ.

Because the method reads as many records as can fit in the buffer, the caller MUST also set one and only one of the following flags to indicate the direction for successive read operations.

Value	Meaning
EVENTLOG_FORWARDS_READ 0x00000004	Log is read in chronological order. This flag MUST NOT be used with EVENTLOG_BACKWARDS_READ.
EVENTLOG_BACKWARDS_READ 0x00000008	Log is read in reverse chronological order. This flag MUST NOT be used with EVENTLOG_FORWARDS_READ.

RecordOffset: Log entry record number from which the read operation should start (this is not a byte offset but a record number). This parameter MUST be ignored unless the EVENTLOG_SEEK_READ bit is set in the *ReadFlags* parameter.

NumberOfBytesToRead: Size of Buffer in bytes. This is the maximum amount of data that can be read.

Buffer: The buffer in which to place data read from the event log.

NumberOfBytesRead: Pointer to a variable that receives the number of bytes actually read by the method.

MinNumberOfBytesNeeded: If the method fails because the buffer is too small to fit even a single record, this MUST be set to the minimum number of bytes needed to fit the next record. Otherwise, this MUST NOT be set, and MUST be ignored by the caller.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success. If the method is successful, the read position MUST be adjusted by NumberOfBytesRead. The method MUST return STATUS_BUFFER_TOO_SMALL (0xC0000023) if the buffer is too small to fit even one record. Otherwise, it MUST return any other implementation-specific non-zero value.

In response to this request from the client, the server MUST check its internal table, and MUST fail the operation if it has no entry in its internal table for the handle.

Then the server MUST determine what record to read next. There are three cases:

1. If the **ReadFlags** field includes the EVENTLOG_SEQUENTIAL_READ flag, and the handle has never been used for reading, the next record MUST be the oldest record in the file when the EVENTLOG_FORWARDS_READ flag is set; or it MUST be the newest record in the file if EVENTLOG_BACKWARDS_READ is set.

2. If the ReadFlags field includes the EVENTLOG_SEQUENTIAL_READ flag, and the handle has been previously used for reading, the handle MUST have the last record read as part of its state. The next record MUST be determined by taking the last record read value from the handle state and either adding one to it (EVENTLOG_FORWARDS_READ is set) or subtracting one from it.

3. If the ReadFlags field includes the EVENTLOG_SEEK_READ flag, the next record to read MUST be specified by the RecordOffset parameter.

If the next record to be read is out of range, the server MUST fail the operation. Examples include the caller trying to seek to a record that does not exist, or the caller requesting a sequential read when all the records have been read.

Once the next record is determined, the server MUST determine how large that record is. If the next record is too large to fit into the buffer, the server MUST fail the method, set the *MinNumberOfBytesNeeded* parameter to the number of bytes needed, and specifically return STATUS_BUFFER_TOO_SMALL (0xC0000023).

If the above checks all succeed, the server MUST attempt to copy as many records as it can into the buffer. The server MUST only copy full event records, and it MUST stop if there are no more events to be read. The server MUST update the handle state to save the record number of the last event copied into the buffer.

The server MUST return a value indicating success or failure for this operation.

3.1.4.8 ElfrReadELA (Opnum 17)

The **ElfrReadELA (Opnum 17)** method reads events from the event log; the server transmits these events to the client and advances the reader's position within the event log associated with this handle. The strings in the returned events MUST be ANSI.

```
NTSTATUS ElfrReadELA(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long ReadFlags,
  [in] unsigned long RecordOffset,
  [in] RULONG NumberOfBytesToRead,
  [out, size_is(NumberOfBytesToRead)]
    unsigned char* Buffer,
  [out] unsigned long* NumberOfBytesRead,
  [out] unsigned long* MinNumberOfBytesNeeded
);
```

LogHandle: Handle to an event log to read. This parameter is an RPC context handle, as specified in [C706] Context Handles.

ReadFlags: The caller MUST specify if the read is to start at a specific record, or is to proceed from the last record read. The value MUST be one and only one of the following flags.

Value	Meaning
EVENTLOG_SEQUENTIAL_READ 0x00000001	Read operation proceeds sequentially from the last call to the **ElfrReadELA** (section 3.1.4.8) method or the **ElfrReadELW (section 3.1.4.7)** method, using this handle. This flag cannot be used with EVENTLOG_SEEK_READ.
EVENTLOG_SEEK_READ 0x00000002	Read operation proceeds from the record specified by the *RecordOffset* parameter. This flag cannot be used with EVENTLOG_SEQUENTIAL_READ.

Because the method reads as many records as can fit in the buffer, the caller MUST also set one and only one of the following flags to indicate the direction for successive read operations.

Value	Meaning
EVENTLOG_FORWARDS_READ 0x00000004	Log is read in chronological order. This flag cannot be used with EVENTLOG_BACKWARDS_READ.
EVENTLOG_BACKWARDS_READ 0x00000008	Log is read in reverse chronological order. This flag cannot be used with EVENTLOG_FORWARDS_READ.

RecordOffset: Log entry record number at which the read operation is to start. Each event in a log has a record number. This parameter MUST be ignored unless the EVENTLOG_SEEK_READ bit is set in the *ReadFlags* parameter.

NumberOfBytesToRead: Size of the buffer in bytes. This is the maximum amount of data that can be read.

Buffer: Data read from the event log.

NumberOfBytesRead: Number of bytes read by the method.

MinNumberOfBytesNeeded: If the method fails because the buffer is too small to fit even a single record, this MUST be set to the minimum number of bytes needed to fit the next record. Otherwise, this MUST NOT be set, and MUST be ignored by the caller.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success. The method MUST return STATUS_BUFFER_TOO_SMALL (0xC0000023) if the buffer is too small to fit even one record. Otherwise, it MUST return any other implementation-specific non-zero value.

This is identical to the **ElfrReadELW** (section 3.1.4.7) method except that the events placed in the buffer MUST be ANSI strings rather than [UNICODE] strings.

3.1.4.9 ElfrClearELFW (Opnum 0)

The **ElfrClearELFW (Opnum 0)** method instructs the server to clear an event log, and, optionally, to back up the event log before the clear operation takes place.

```
NTSTATUS ElfrClearELFW(
  [in] IELF_HANDLE LogHandle,
  [in, unique] PRPC_UNICODE_STRING BackupFileName
);
```

LogHandle: Handle to the event log to be cleared. This parameter is an RPC context handle, as specified in [C706]. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the **ElfrOpenBELW (section 3.1.4.1)** method.

BackupFileName: NT Object Path of a file in which a current copy of the event log is to be placed. If this is NULL or empty, no backup is to be created. The path is relative to the server rather than the client.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

In response to this request from the client, the server SHOULD first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle, or if the handle is for a backup event log. Handles to backup event logs are obtained via the **ElfrOpenBELW** (section 3.1.4.1) method or the **ElfrOpenBELA** (section 3.1.4.2) method<21>.

If the *BackupFileName* is non-NULL and non-empty, the server MUST validate the *BackupFileName* and fail the call if it is not a valid name. An invalid name is defined as an illegal NT Object Path or a path that specifies a file that already exists. If the *BackupFileName* is valid, the server MUST attempt to back up the log to the path specified in *BackupFileName* before the log is cleared. The method MUST fail the operation and not clear the log if the user does not have write access to the location specified by the BackupFileName parameter of if the backup does not succeed for any other reason.

If the *BackupFileName* is NULL or empty, the method MUST NOT attempt to back up the event log.

If the preceding checks are successful, and if no problems occur during creation of a backup log, the server MUST attempt to clear the associated event log. All events MUST be removed during clearing. Additionally, the state of the log MUST be modified so that the next record written has a record number 1. The server MUST return a value indicating success or failure for this operation.

3.1.4.10 ElfrClearELFA (Opnum 12)

The **ElfrClearELFA (Opnum 12)** method instructs the server to clear an event log, and, optionally, to back up the event log before the clear operation takes place.

```
NTSTATUS ElfrClearELFA(
  [in] IELF_HANDLE LogHandle,
  [in, unique] PRPC_STRING BackupFileName
);
```

LogHandle: Handle to the event log to be cleared. This parameter is an RPC context handle, as specified in [C706], Context Handles. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the **ElfrOpenBELW (section 3.1.4.1)** method.

BackupFileName: NT Object Path of a file (as specified in section 2.2.6.1), in which a current copy of the event log is to be placed. If this is NULL or empty, the server MUST NOT create a backup as part of this method.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

The **ElfrClearELFA** and **ElfrClearELFW (section 3.1.4.9)** methods are identical in functionality. The difference between the two methods is that the **ElfrClearELFA** method specifies *BackupFileName* as an ANSI string. The **ElfrClearELFW** method specifies *BackupFileName* as a UNICODE string.

For more information, see the Remarks section of the **ElfrClearELFW** method.

3.1.4.11 ElfrBackupELFW (Opnum 1)

The **ElfrBackupELFW (Opnum 1)** method instructs the server to back up the event log to a specified file name.

```
NTSTATUS ElfrBackupELFW(
  [in] IELF_HANDLE LogHandle,
  [in] PRPC_UNICODE_STRING BackupFileName
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706]. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the **ElfrOpenBELW (section 3.1.4.1)** method.

BackupFileName: NT Object Path of a file, (as specified in section 2.2.6.1), in which a current copy of the event log is to be placed. This MUST NOT be NULL or empty. The path is evalutated relative to the server.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle, or if the handle is for a backup event log. Handles to backup event logs are obtained via the **ElfrOpenBELW** (section 3.1.4.1) method or the **ElfrOpenBELA** (section 3.1.4.2) method.

If the handle is valid, the server MUST validate the *BackupFileName* and fail the call if it is not a legal NT Object Path, or if it specifies a file that already exists.

If the checks above are successful, the server MUST attempt to create a backup of the log associated with the *LogHandle* parameter and return a value indicating success or failure for this operation.

3.1.4.12 ElfrBackupELFA (Opnum 13)

The **ElfrBackupELFA (Opnum 13)** method instructs the server to back up the event log to a specified file name.

```
NTSTATUS ElfrBackupELFA(
  [in] IELF_HANDLE LogHandle,
  [in] PRPC_STRING BackupFileName
```

```
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

BackupFileName: NT Object Path of a file (as specified in section 2.2.6.1), in which a current copy of the event log is to be placed. This MUST NOT be NULL or empty.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

This is identical to the **ElfrBackupELFW (section 3.1.4.11)** method except that BackupFileName is an ANSI string in this case.

3.1.4.13 ElfrReportEventW (Opnum 11)

The **ElfrReportEventW (Opnum 11)** method writes events to the event log; the server receives these events from the client.

```
NTSTATUS ElfrReportEventW(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long Time,
  [in] unsigned short EventType,
  [in] unsigned short EventCategory,
  [in] unsigned long EventID,
  [in, range(0, 256)] unsigned short NumStrings,
  [in, range(0, 61440)] unsigned long DataSize,
  [in] PRPC_UNICODE_STRING ComputerName,
  [in, unique] PRPC_SID UserSID,
  [in, size_is(NumStrings), unique]
    PRPC_UNICODE_STRING Strings[*],
  [in, size_is(DataSize), unique]
    unsigned char* Data,
  [in] unsigned short Flags,
  [in, out, unique] unsigned long* RecordNumber,
  [in, out, unique] unsigned long* TimeWritten
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the ElfrOpenBELW (section 3.1.4.1) method.

Time: Time at which the event was generated by the event source (not the time at which the event was logged). The time MUST be expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC).

EventType: Type of the event, as specified in section 2.2.4.

EventCategory: Event category, as specified in section 1.8.5.

EventID: EventID, as specified in section 3.1.1.4.

NumStrings: Number of strings in the array pointed to by the Strings parameter. A value of 0 indicates that no strings are present.

DataSize: Number of bytes of event-specific raw binary data to write to the log. This binary data is passed in the *Data* parameter. If the *DataSize* parameter is 0, event-specific data MUST NOT be present.

ComputerName: A string to assist in identifying the machine that generated the event. In practice, the name of the computer. There are no character restrictions on this field's content (for example, a fully-qualified DNS name can be used)<22>.

UserSID: Either NULL or a user SID. If this is NULL, the event is to have a 0 length **UserSid** field.

Strings: Specifies strings containing information specific to the event. This parameter MUST be a valid pointer. If the *NumStrings* parameter is 0, this parameter MUST be NULL. For example, an event relating to file deletion could use a string to specify the path of the file being deleted.

Data: Pointer to the buffer that contains the event-specific binary data. This parameter MUST be a valid pointer (or NULL), even if the *DataSize* parameter is 0.

Flags: Unused. MUST be set to 0, and ignored on receipt.

RecordNumber: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

TimeWritten: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid.

If the handle is valid, the method MUST attempt to create an event with the supplied parameters and by setting the **TimeWritten** and the **RecordNumber** fields in the event. The **TimeWritten** MUST be obtained from the system clock. The server MUST get the RecordNumber from the state maintained for the event log. The server SHOULD set the **TimeWritten** and **RecordNumber** parameters to the same values written to the event prior to returning from this method<23>.

The server MUST ignore the *TimeWritten* and *RecordNumber* parameters received from the client.

Then the server MUST attempt to store the event source name in the event. This event source was originally specified when the **ElfrRegisterEventSourceW (section 3.1.4.5)** method or the **ElfrRegisterEventSourceA (section 3.1.4.6)** method was called.

If the above checks all succeed, the server MUST attempt to copy the event into the event log and attempt to update the log state so that the record number is incremented for the next write.

The server MUST return a value indicating success or failure for this operation.

3.1.4.14 ElfrReportEventA (Opnum 18)

The **ElfrReportEventA (Opnum 18)** method writes events to the event log; the server receives these events from the client.

```
NTSTATUS ElfrReportEventA(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long Time,
  [in] unsigned short EventType,
  [in] unsigned short EventCategory,
  [in] unsigned long EventID,
  [in, range(0, 256)] unsigned short NumStrings,
  [in, range(0, 61440)] unsigned long DataSize,
  [in] PRPC_STRING ComputerName,
  [in, unique] PRPC_SID UserSID,
  [in, size_is(NumStrings), unique]
    PRPC_STRING Strings[*],
  [in, size_is(DataSize), unique]
    unsigned char* Data,
  [in] unsigned short Flags,
  [in, out, unique] unsigned long* RecordNumber,
  [in, out, unique] unsigned long* TimeWritten
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the **ElfrOpenBELW (section 3.1.4.1)** method.

Time: Time at which the event was generated by the event source (not the time at which the event was). The time MUST be expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC).

EventType: Type of the event, as specified in section 2.2.4.

EventCategory: Event category, as specified in section 1.8.5.

EventID: EventID, as specified in section 3.1.1.4.

NumStrings: Number of strings in the array pointed to by the *Strings* parameter. A value of 0 indicates that no strings are present.

DataSize: Number of bytes of event-specific raw binary data to write to the log. This binary data is passed in the *Data* parameter. If no event-specific data is present, this parameter MUST be set to 0.

ComputerName: A string to assist in identifying the machine that generated the event. In practice, the name of the computer. There are no character restrictions on this field's content (for example, a fully-qualified DNS name can be used)<24>.

UserSID: Either NULL or a user SID. If this is NULL, the event is to have a 0 length **UserSid** field.

Strings: Specifies strings containing information specific to the event. This parameter MUST be a valid pointer. If the *NumStrings* parameter is 0, this parameter MUST be NULL. For example, an event relating to file deletion could use a string to specify the path of the file being deleted.

Data: Pointer to the buffer that contains the event-specific binary data. This parameter MUST be a valid pointer (or NULL), even if the *DataSize* parameter is 0.

Flags: Unused. MUST be set to 0, and ignored on receipt.

RecordNumber: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

TimeWritten: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

This method is identical to the **ElfrReportEventW (section 3.1.4.13)** method except that the string arguments are ANSI strings in this case. Thus, the remarks in **ElfrReportEventW** (section 3.1.4.13) apply to this method as well.

3.1.4.15 ElfrReportEventAndSourceW (Opnum 24)

This method<25> instructs the server to write an event to an event log. It differs from the other methods for writing an event by specifying the event source at the time of the write. The other methods for writing an event required the event source to be specified when the handle was opened for write. This method is intended for client applications that forward events, reading them from one event log and writing them to another. Such applications need to forward the events from multiple original event sources.

```
NTSTATUS ElfrReportEventAndSourceW(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long Time,
  [in] unsigned short EventType,
  [in] unsigned short EventCategory,
  [in] unsigned long EventID,
  [in] PRPC_UNICODE_STRING SourceName,
  [in, range(0, 256)] unsigned short NumStrings,
  [in, range(0, 61440)] unsigned long DataSize,
  [in] PRPC_UNICODE_STRING ComputerName,
  [in, unique] PRPC_SID UserSID,
  [in, size_is(NumStrings), unique]
    PRPC_UNICODE_STRING Strings[*],
  [in, size_is(DataSize), unique]
    char* Data,
  [in] unsigned short Flags,
  [in, out, unique] unsigned long* RecordNumber,
  [in, out, unique] unsigned long* TimeWritten
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles. This handle MUST NOT be one obtained via the **ElfrOpenBELA (section 3.1.4.2)** method or the **ElfrOpenBELW (section 3.1.4.1)** method.

Time: Time at which the event was generated by the event source (not the time at which the event was logged). The time MUST be expressed as the number of seconds since 00:00:00 on January 1, 1970 (UTC).

EventType: Type of the event, as specified in section 2.2.4.

EventCategory: Event category, as specified in section 1.8.5.

EventID: EventID, as specified in section 3.1.1.4.

SourceName: Specifies the name of the event source.

NumStrings: Number of strings in the array pointed to by the *Strings* parameter. If no strings are present, this value MUST be set to 0.

DataSize: Number of bytes of event-specific raw binary data to write to the log. This binary data is passed in the *Data* parameter. If no event-specific data is present, this parameter MUST be set to 0.

ComputerName: A string to assist in identifying the machine that generated the event. In practice, the name of the computer. There are no character restrictions on this field's content (for example, a fully-qualified DNS name can be used).

UserSID: Either NULL or a user SID. If this is NULL, the event is to have a 0 length **UserSid** field.

Strings: Strings containing text information specific to the event. This parameter MUST be a valid pointer. If the *NumStrings* parameter is 0, this parameter MUST be NULL. For example, an event relating to file deletion could use a string to specify the path of the file being deleted.

Data: Pointer to a buffer that contains binary information specific to the event. This parameter MUST be a valid pointer (or NULL), even if the DataSize parameter is 0.

Flags: Indicates if the buffer (pointed to by the *Data* parameter) is formatted as XML.

Value	Meaning
0x00000000	The event is not formatted as XML.
0x00008000	The event is formatted as XML.

RecordNumber: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

TimeWritten: Unused. Can be set to any arbitrary value, and any value sent by the client MUST be ignored by the server.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a non-zero value.

Note If the method is not supported, the RPC transport itself (as opposed to this protocol) returns RPC_S_PROCNUM_OUT_OF_RANGE (0x6D1).

This method is almost identical to the **ElfrReportEventW (section 3.1.4.13)** method except for the following difference: In this method, the server MUST get the event source name from the *SourceName* parameter rather than from any value stored with the handle.

The values for all returned data, including error codes and out parameters, are contained in the **stub_data** field of the RPC_RESPONSE packet.

3.1.4.16 ElfrNumberOfRecords (Opnum 4)

The **ElfrNumberOfRecords (Opnum 4)** method instructs the server to report the number of records currently in the event log.

```
NTSTATUS ElfrNumberOfRecords(
  [in] IELF_HANDLE LogHandle,
  [out] unsigned long* NumberOfRecords
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

NumberOfRecords: Total number of records in the specified event log.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid.

If the handle is valid, the method MUST retrieve the number of records in the associated log and return the number via the *NumberOfRecords* parameter, and return success. This call MUST NOT update the internal state of the server.

3.1.4.17 ElfrOldestRecord (Opnum 5)

The **ElfrOldestRecord (Opnum 5)** method instructs the server to report the record number of the oldest record in the event log.

```
NTSTATUS ElfrOldestRecord(
  [in] IELF_HANDLE LogHandle,
  [out] unsigned long* OldestRecordNumber
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

OldestRecordNumber: The number of the oldest record in the specified event log. The chronology is based on the time that records are written (not the record generation time specified by the event source).

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return a nonzero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid.

If the handle is valid, the method MUST retrieve the record number of the oldest record in the associated log and return the number via the *OldestRecordNumber* parameter, and return success. If the log is empty, the server MUST set the *OldestRecordNumber* parameter to 0. This call MUST NOT update the internal state of the server.

3.1.4.18 ElfrGetLogInformation (Opnum 22)

The **ElfrGetLogInformation (Opnum 22)** method instructs the server to return information on an event log.

```
NTSTATUS ElfrGetLogInformation(
  [in] IELF_HANDLE LogHandle,
  [in] unsigned long InfoLevel,
  [out, size_is(cbBufSize)] unsigned char* lpBuffer,
  [in, range(0, 1024)] unsigned long cbBufSize,
  [out] unsigned long* pcbBytesNeeded
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

InfoLevel: The level of event log information to return. This MUST be set to 0.

lpBuffer: The event log information. This MUST point to either an **EVENTLOG_FULL_INFORMATION (section 2.2.6)** structure or be NULL.

cbBufSize: The size in bytes of the buffer pointed to by the *lpBuffer* parameter.

pcbBytesNeeded: Number of bytes required for the requested information, regardless of if the function succeeds. This parameter MUST NOT be NULL.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success. The method MUST return STATUS_BUFFER_TOO_SMALL (0xC0000023) if the buffer is too small to fit even one record. Otherwise, it MUST return any other implementation-specific non-zero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid.

If lpBuffer is not large enough to contain an **EVENTLOG_FULL_INFORMATION** (section 2.2.6) structure, the server MUST set the *pcbBytesNeeded* parameter to the number of bytes needed to hold an **EVENTLOG_FULL_INFORMATION** (section 2.2.6) structure, MUST fail the method, and MUST return STATUS_BUFFER_TOO_SMALL (0xC0000023).

If the above checks all succeed, the server MUST fill in an **EVENTLOG_FULL_INFORMATION** (section 2.2.6) structure into the lpBuffer with the dwFull member of the structure being set to 1 if the event log is full, and set to 0 if the event log is not full. In addition, the *pcbBytesNeeded* parameter MUST be set to the size of an **EVENTLOG_FULL_INFORMATION** (section 2.2.6) structure. The server MUST then return success.

3.1.4.19 ElfrCloseEL (Opnum 2)

The **ElfrCloseEL (Opnum 2)** method instructs the server to close a handle to the event log, freeing resources on the server that maintained an association between the handle and the corresponding event log. This handle MUST have been obtained via the **ElfrOpenELW (section 3.1.4.3)** method, the **ElfrOpenELA (section 3.1.4.4)** method, the **ElfrOpenBELW (section 3.1.4.1)** method, or the **ElfrOpenBELA (section 3.1.4.2)** method.

```
NTSTATUS ElfrCloseEL(
  [in, out] IELF_HANDLE* LogHandle
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706] Context Handles.

Return Values: The method MUST return STATUS_SUCCESS (0x00000000) on success; otherwise, it MUST return an implementation-specific nonzero value.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid. The server MUST treat a handle opened by the **ElfrOpenELW** (section 3.1.4.3) method, the **ElfrOpenELA** (section 3.1.4.4) method, the **ElfrOpenBELW** (section 3.1.4.1) method, or the **ElfrOpenBELA** (section 3.1.4.2) method as valid.

If the handle is valid, the server MUST remove the handle to the log from its internal table and return success.

3.1.4.20 ElfrDeregisterEventSource (Opnum 3)

The **ElfrDeregisterEventSource (Opnum 3)** method instructs the server to close a handle to the event log, freeing resources on the server that maintained an association between the handle and the corresponding event log. This handle MUST have been obtained via the **ElfrRegisterEventSourceW (section 3.1.4.5)** method or the **ElfrRegisterEventSourceA (section 3.1.4.6)** method.

```
NTSTATUS ElfrDeregisterEventSource(
  [in, out] IELF_HANDLE* LogHandle
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

Return Values: The method MUST return STATUS_SUCCESS on success; otherwise, it MUST return a nonzero value.

Note STATUS_SUCCESS has a value of 0x00000000.

In response to this request from the client, the server MUST first check that the handle exists in its internal table. The server MUST fail the operation if it has no entry in its internal table for the handle or if the handle is not valid. The server MUST treat a handle opened by the **ElfrRegisterEventSourceW** (section 3.1.4.5) method or the **ElfrRegisterEventSourceA** (section 3.1.4.6) method as valid.

If the handle is valid, the server MUST remove the handle to the log from its internal table and return success.

3.1.4.21 ElfrChangeNotify (Opnum 6)

The **ElfrChangeNotify (Opnum 6)** method is intended for local use.

```
NTSTATUS ElfrChangeNotify(
  [in] IELF_HANDLE* LogHandle,
  [in] RPC_CLIENT_ID ClientId,
  [in] unsigned long Event
);
```

LogHandle: Handle to an event log. This parameter is an RPC context handle, as specified in [C706], Context Handles.

ClientId: Ignored when the method is called remotely.

Event: Ignored when the method is called remotely.

Return Values: The method always returns an error value when called remotely. <26>

In response to this request from the client, the server MUST first validate the handle and return an error code <27> if the handle is remote.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 Client Details

The client side of this RPC protocol is simply a pass-through.

The client implementation also relies on a combination of remote registry (as specified in [MS-RRP]), remote SMB, LSA, and LDAP protocols to retrieve and assemble event description string. For these operations, see section 3.2.4.1.

3.2.1 Abstract Data Model

The client does not maintain state as part of the EventLog Remoting Protocol.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Message Processing Events and Sequencing Rules

Calls made by the higher-layer protocol or application are passed directly to the transport. All return values from method invocations MUST be returned uninterpreted to the higher-layer protocol or application.

Methods in RPC Opnum Order

Method	Description
ElfrClearELFW	This method instructs the server to clear an event log, and, optionally, to back up the event log before the clear takes place. Opnum: 0
ElfrBackupELFW	This method instructs the server to back up the event log to a specified file name. Opnum: 1
ElfrCloseEL	This method instructs the server to close a handle to the event log, freeing resources on the server that maintained an association between the handle and the corresponding event log. This handle MUST have been obtained via the **ElfrOpenELW (section 3.1.4.3)** method, **ElfrOpenELA (section 3.1.4.4)** method, **ElfrOpenBELW (section 3.1.4.1)** method, or **ElfrOpenBELA (section 3.1.4.2)** method. Opnum: 2
ElfrDeregisterEventSource	This method instructs the server to close a handle to the event log, freeing resources on the server that maintained an association between the handle and the corresponding event log. This handle MUST have been obtained via the **ElfrRegisterEventSourceW (section 3.1.4.5)** method or the **ElfrRegisterEventSourceA (section 3.1.4.6)** method. Opnum: 3
ElfrNumberOfRecords	This method instructs the server to report the number of records currently in the event log. Opnum: 4
ElfrOldestRecord	This method instructs the server to report the record number of the oldest record in the event log. Opnum: 5
ElfrChangeNotify	Reserved for local use. Notifies local processes about changes to the event log. Opnum: 6
ElfrOpenELW	This method instructs the server to return a handle to a live event log. The caller MUST<28> have permission to read the file containing the event log for this to succeed. Opnum: 7
ElfrRegisterEventSourceW	This method instructs the server to return a handle to an event log for writing. The caller MUST have permission to write to the file containing

Method	Description
	the event log for this to succeed. The module name argument specifies the event sources, which is used to determine the relevant event log, as specified below. Opnum: 8
ElfrOpenBELW	This method instructs the server to return a handle to a backup event log. The caller MUST have permission to read the file containing the backup event log for this to succeed. Opnum: 9
ElfrReadELW	This method reads events from the event log; the server transmits these events to the client and advances the reader's position within the event log associated with this handle. The strings in the returned event MUST be in [UNICODE]. Opnum: 10
ElfrReportEventW	This method writes events to the event log; the server receives these events from the client. Opnum: 11
ElfrClearELFA	This method instructs the server to clear an event log, and, optionally, to back up the event log before the clear takes place. Opnum: 12
ElfrBackupELFA	This method instructs the server to back up the event log to a specified file name. Opnum: 13
ElfrOpenELA	This method instructs the server to return a handle to a live event log. The caller MUST<29>have permission to read the file containing the event log for this to succeed. Opnum: 14
ElfrRegisterEventSourceA	This method instructs the server to return a handle to an event log for writing. The caller MUST have permission to write to the file containing the event log for this to succeed. The module name argument specifies the event source, which is used to determine the relevant event log, as specified in the following sections. Opnum: 15
ElfrOpenBELA	This method instructs the server to return a handle to a backup event log. The caller MUST have permission to read the file containing the backup event log for this to succeed. Opnum: 16
ElfrReadELA	This method reads events from the event log; the server transmits these events to the client and advances the reader's position within the event log associated with this handle. The strings in the returned events MUST be ANSI. Opnum: 17
ElfrReportEventA	This method writes events to the event log; the server receives these events from the client. Opnum: 18

Method	Description
Opnum19NotUsedOnWire	Reserved for local use. Opnum: 19
Opnum20NotUsedOnWire	Reserved for local use. Opnum: 20
Opnum21NotUsedOnWire	Reserved for local use. Opnum: 21
ElfrGetLogInformation	This method instructs the server to return information on an event log. Opnum: 22
Opnum23NotUsedOnWire	Reserved for local use. Opnum: 23
ElfrReportEventAndSourceW	This method<30> instructs the server to write an event to an event log. It differs from the other methods for writing an event by specifying the event source at the time of the write. The other methods for writing an event required the event source to be specified when the handle was opened for write. This method is intended for client applications that forward events, reading them from one event log and writing them to another. Such applications need to forward the events from multiple original event sources. Opnum: 24

In the preceding table, the phrase "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined because it does not affect interoperability.

3.2.4.1 Client Processing of Event Descriptions and Other Localizable Strings

The server MAY store various localizable description strings in resource binary files formatted according to the Microsoft Portable Executable and Common Object File Format Specification, as specified in [PE-COFF]. These resource files MAY be specified per event log or per event source. When the resource files are created for event logs the server MUST add paths to their locations into the registry using the Windows Remote Registry Protocol, as specified in [MS-RRP], as registry values under each log's registry location described in 3.1.1.2. When the resource files are created for event sources, the server MUST add paths to their locations into the registry using the Windows Remote Registry Protocol, as specified in [MS-RRP], as registry values under each source's registry location described in section 3.1.1.3.

3.2.4.1.1 Loading Event Log Description Information

If the log contains a localizable display name, the server MUST specify it via "DisplayNameFile" and "DisplayNameID" registry values under the log registry key described in section 3.1.1.2.

"DisplayNameFile" value MUST be of type REG_EXPAND_SZ.

"DisplayNameID" value MUST of type REG_DWORD.

"DisplayNameFile" data MAY contain environment variables enclosed by percent (%). The client MUST attempt to expand an environment variable, as described in section 3.2.4.1.5.4 to retrieve the full path to the resource file. If the client is accessing a remote source, it MUST then convert the expanded resource file path to a UNC path: When the path begins with "X:" pattern, where the first

character is a drive letter and the second character is ":", the client MUST transform it to \\messageSourceServer\X$\path.

The format of the resource file is specified in [PE-COFF].

When both "DisplayNameFile"and "DisplayNameID" values are present, the client SHOULD attempt to load the resource file<31> by using the Server Message Block (SMB) Protocol, as specified in [MS-SMB], and to retrieve the resource string with the ID number specified by the "DisplayNameID" value data <32>.

3.2.4.1.2 Retrieving Event Parameter Strings

If the event source contains parameter strings, the server MUST specify a "ParameterMessageFile" registry value under the source registry key specified in section 3.1.1.3.

"ParameterMessageFile" value MUST be of type REG_EXPAND_SZ.

"ParameterMessageFile" data MAY contain environment variables enclosed by % signs. The client MUST attempt to expand an environment variable as specified in section 3.2.4.1.5.4 to retrieve the full path to the resource file. If the client is accessing a remote source, it MUST then convert the expanded resource file path to a UNC path: when the path begins with an "X:" pattern, where the first character is a drive letter and the second character is ":", the client MUST transform it to \\messageSourceServer\X$\path.

The format of the resource file is specified in [PE-COFF].

When "ParameterMessageFile" value is present, the client SHOULD attempt to load the resource file<33> using the SMB protocol, as specified in [MS-SMB], and retrieve parameter resource strings with ID numbers corresponding to the parameter insertion code encountered during the process of expanding a description string for that source.<34> Parameter insertion rules are further described in 3.2.4.1.5.2.

3.2.4.1.3 Retrieving Event Category Strings

The server MAY specify a **Category Message File** for an event source—a binary resource file defining description strings for event categories, where the resource ID corresponds to the category number. Thus, string 2 is the **category string** for all EventIDs for this source of category 2.

If an event source contains localizable category names, the server MUST specify them via "CategoryMessageFile" and "CategoryCount" registry values under the log registry key described in 3.1.1.3.

"CategoryMessageFile " value MUST be of type REG_EXPAND_SZ.

"CategoryCount " value MUST of type REG_DWORD.

"CategoryMessageFile "data MUST contain a single path to a category message file for this source. The path data MAY contain environment variables enclosed by % signs. The client MUST attempt to expand an environment variable as described in 3.2.4.1.5.4 to retrieve the full path to the resource file. If the client is accessing a remote source, it MUST then convert the expanded resource file path to a UNC path; when the path begins with an "X:" pattern, where the first character is a drive letter and the second character is ":", the client MUST transform it to \\messageSourceServer\X$\path.<35>

The format of the resource file is specified in [PE-COFF].

"CategoryCount" is the number of categories for this event source. Note that, unlike Event IDs and parameters, category numbers are required to be be sequential starting from 1.

When both "CategoryMessageFile "and "CategoryCount " values are present, the client SHOULD attempt to load the category resource file<36> using the SMB protocol, as specified in [MS-SMB], and retrieve the category resource string with ID number specified by the "EventCategory" EVENTLOGRECORD field.<37> The client MAY retrieve all category descriptions at once and cache them for subsequent access.

3.2.4.1.4 Retrieving Unexpanded Event Description Strings

If the event source contains localizable event description strings, the server MUST specify an "EventMessageFile" registry value under the source registry key described in 3.1.1.3.

The Event Message File is a binary resource file defining **unexpanded description strings** for an event source, where the resource ID corresponds to the EventID. Thus, a string with the resource ID 5 is the unexpanded description string for events with EventID 5, where EventID is a field of EVENTLOGRECORD as specified in 2.2.5.

"EventMessageFile" value MUST be of type REG_EXPAND_SZ.

"EventMessageFile" data MAY contain environment variables enclosed by % signs. The client MUST attempt to expand an environment variable as specified in 3.2.4.1.5.4 to retrieve the full path to the resource file. If the client is accessing a remote source, it MUST then convert the expanded resource file path to a UNC path: when the path begins with an "X:" pattern, where the first character is a drive letter and the second character is ":", the client MUST transform it to \\messageSourceServer\X$\path.

The format of the resource file is specified in [PE-COFF].

"EventMessageFile" data MAY contain several paths to **event message files** for this source, delimited by comma or semicolon. The client MUST expand any environment variables in each file path as specified above.

When an "EventMessageFile" value is present, the client SHOULD attempt to load the resource file<38> using the SMB protocol, as specified in [MS-SMB], and retrieve the unexpanded description string with a resource ID number corresponding to the EventID for that record. <39>

If several event message files are specified, the client MUST attempt to load the resource string from these files in the order in which the files are specified until the resource string is successfully loaded.

The server MAY specify a "PrimaryModule" REG_EXPAND_SZ registry value under the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\<LogName> registry key. If the client cannot find an event description string, it SHOULD attempt to load the "PrimaryModule" value for the event log. The client SHOULD use the file whose path is found in the "PrimaryModule" value as a fallback message file for loading event description strings all sources in the log. "PrimaryModule" data MAY contain environment variables enclosed by % signs. The client MUST attempt to expand an environment variable as specified in 3.2.4.1.5.4 to retrieve the full path to the resource file. If the client is accessing a remote source, it MUST then convert the expanded resource file path to a UNC path : when the path begins with an "X:" pattern, where the first character is a drive letter and the second character is ":", the client MUST transform it to \\messageSourceServer\X$\path.

3.2.4.1.5 Expanding Unexpanded Event Description Strings

The following insertion codes MAY appear as part of unexpanded description strings or during the process of expanding a description string.

Insertion code	Explanation
%n	EVENTLOGRECORD string: string indexed (n -1) in the EVENTLOGRECORD.Strings array, as specified in section 2.2.5
%%n	Parameter string: string with resource ID n in the **parameter message file**
%{S...}	Security Identifier (SID)
%{...}	**Globally unique identifier (GUID)**

Client MUST replace insertion codes recursively,<40> so that if the expanded string appears to contain another insertion code, it will also be expanded. Client MUST limit the number of substitutions to ensure that the algorithm will complete.<41>

This protocol does not contain an "escaping" mechanism to allow a literal string to go unexpanded when it contains a substring that looks like an insertion code. For example, suppose the unexpanded description string contains insertion code "%1", and the first EVENTLOGRECORD string (as specified in section 2.2.5) is a file name that in this case happens to contain "%1". The filename will be substituted into the description string, then the "%1" inside the filename will be interpreted as an insertion code and substituted with the whole filename, and so on, until the cap on the number of substitutions is reached (if such a cap is part of the client implementation).

3.2.4.1.5.1 Inserting EVENTLOGRECORD Strings

Individual EVENTLOGRECORD structures, as defined in 2.2.5, can have zero or more strings attached to them. The first EVENTLOGRECORD string MUST be specified in the unexpanded description string as "%1", not "%0".

For example, when the client encounters insertion code "%1" within an unexpanded description string, it MUST replace it with the first attached string.

If EVENTLOGRECORD string insertion fails for any reason, the client SHOULD quote the insertion code verbatim.

3.2.4.1.5.2 Inserting Parameter Strings

See 3.2.4.1.2 for how parameter strings are retrieved.

When the client encounters insertion code "%%n", it MUST attempt to replace the insertion code with resource ID number n from the parameter message file.

If parameter replacement fails for any reason, the client SHOULD quote the insertion code verbatim.

3.2.4.1.5.3 Inserting SIDs and GUIDs

Services that write Event Log entries MAY record the identity of Active Directory objects (for example, users) and security principals as insertion strings in (respectively) GUID and SID format. The client MAY expand these GUIDs and SIDs into readable names of objects.<42>

If the client does not attempt to resolve SID or GUID insertion codes, or if resolving them fails for any reason, the client SHOULD quote SID and GUID strings verbatim.

3.2.4.1.5.4 Expanding Environment Variables

Whenever the server registry contains a path to a message file, that (REG_EXPAND_SZ) path MAY contain a reference to an environment variable such as "%systemroot%" or "%systemdrive%".

Clients MUST attempt to expand "%systemroot%" and "%systemdrive%" environment variables. Client MUST attempt to replace "%systemroot%" with the registry value on the server computer read in key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion" value "SystemRoot", and "%systemdrive%" with the first two characters of that value <43>.

Expanding environment variables other than "%systemroot%" and "%systemdrive%" is not part of this protocol. The client MAY employ other heuristics or leave these variables unexpanded.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

4 Protocol Examples

4.1 Obtain Records Stored in an Event Log

In this example, a client application wants to obtain records stored in an event log. This involves the following steps:

1. To establish a connection to the server, the client application calls **ElfrOpenELW** with the following values for the parameters.

```
NTSTATUS = {to be filled in by server}
ElfrOpenELW(
  [in] EVENTLOG_HANDLE_W
    UNCServerName = "servername",
  [in] PRPC_UNICODE_STRING ModuleName = {"Application"},
  [in] PRPC_UNICODE_STRING RegModuleName = {""},
  [in] unsigned long MajorVersion = 0x00000001,
  [in] unsigned long MinorVersion = 0x00000001,
  [out] IELF_HANDLE * LogHandle =
    {to be filled in by server}
);
```

   The server verifies that the client application has read access, and, if so, returns a handle (LogHandle) to the client application. The server maintains an association between the handle and a particular event log as well as keeping track of the position of the last read operation, if any.

2. The client application then reads the records using the **ElfrReadELW (section 3.1.4.7)** method. The client application specifies the context handle (LogHandle) obtained in the previous step. To retrieve records in sequential order, the client application calls **ElfrReadELW** with the following parameters:

```
NTSTATUS = {to be filled in by server}
ElfrReadELW(
[in] IELF_HANDLE LogHandle =
{handle obtained by the call to ElfrOpenELW},
[in] unsigned long ReadFlags = 0x00000005,
[in] unsigned long RecordOffset = 0x00000000,
[in, range(0, 0x7FFFF)]
unsigned long NumberOfBytesToRead = 0x3ffff,
[out, size_is(NumberOfBytesToRead)]
unsigned char * Buffer = {to be filled in by server},
[out] unsigned long * NumberOfBytesRead =
{to be filled in by server},
[out] unsigned long * MinNumberOfBytesNeeded =
{to be filled in by server}
);
```

   The server then returns one or more records. The number of records returned is limited by what fits in the buffer and by what is actually available in the log, whatever is less.

3. The client application may continue invoking this method to obtain additional records. When the client application is finished reading records, it releases the log handle by calling **ElfrCloseEL** as follows:

```
NTSTATUS = {to be filled in by server} ElfrCloseEL(
[in, out] IELF_HANDLE * LogHandle
= {handle obtained from ElfrOpenELW}
);
```

The server removes its state for the handle and returns success.

4.2 Write Events to an Event Log

In this example, the client application wants to write events to an event log. This involves the following:

1. To establish a connection to the server, the client application calls **ElfrRegisterEventSourceW** with the following values for the parameters:

```
NTSTATUS = {to be filled in by server}
   ElfrRegisterEventSourceW(
     [in] EVENTLOG_HANDLE_W
        UNCServerName = "servername",
     [in] PRPC_UNICODE_STRING ModuleName = {"Application"},
     [in] PRPC_UNICODE_STRING RegModuleName = {""},
     [in] unsigned long MajorVersion = 0x00000001,
     [in] unsigned long MinorVersion = 0x00000001,
     [out] IELF_HANDLE * LogHandle =
        {to be filled in by server}
     );
```

The server verifies that the client application has write access, and, because the client application has write access in this example, the server returns a handle to the client application. The server maintains an association between the handle and a particular event log.

2. The client application writes the events by using the **ElfrReportEventW** method. The client application specifies the context handle (LogHandle) obtained in the preceding step. The parameters to the **ElfrReportEventW** method are as follows:

```
NTSTATUS = {to be filled in by server}
ElfrReportEventW(
[in] IELF_HANDLE LogHandle =
{handle obtained from ElfrRegisterEventSourceW},
[in] unsigned long Time =
{number of seconds since Jan 1, 1970},
[in] unsigned short EventType = 0x0001,
[in] unsigned short EventCategory = 0x0000,
[in] unsigned long EventID = 0x00000017,
[in, range(0, 0x100)]
unsigned short NumStrings = 0x0002,
[in, range(0, 0x3FFFF)]
```

```
unsigned long DataSize = 0x00000000,
[in] PRPC_UNICODE_STRING
ComputerName = {"MyComputer"},
[in, unique] PRPC_SID UserSID = NULL,
[in, size_is(NumStrings), unique]
PRPC_UNICODE_STRING Strings[*] = {"hello" "world"},
[in, size_is(DataSize), unique]
unsigned char * Data = NULL,
[in] unsigned short Flags = 0x0000,
[in, out, unique] unsigned long * RecordNumber = NULL,
[in, out, unique] unsigned long * TimeWritten = NULL
);
```

The server writes a record to the event log that is associated with the handle and returns success.

The client application may continue invoking this method to write additional events.

3. When the client application is finished writing events, it releases the handle by calling **ElfrDeregisterEventSource** with the following parameter:

```
NTSTATUS = {to be filled in by server}
ElfrDeregisterEventSource(
  [in, out] IELF_HANDLE * LogHandle =
   {handle obtained from ElfrRegisterEventSourceW}
);
```

The server removes its state for the handle and returns success.

4.3 Expanding Unexpanded Event Description Strings

In this example, assume that the unexpanded description string is "Error %2 occurred while performing operation %1 on file %3", and that the record strings (as specified in section 2.2.5) are:

```
first String:        "%%2"
second String:       "Access Denied"
third String:        "C:\securestuff\noaccess.db"
parameter string 2:  "Modify"
```

The first substitution replaces "%2" with "Access Denied", resulting in:

```
Error Access Denied occurred while performing operation %1 on file %3
```

The next substitution replaces "%1" with "%%2", resulting in:

```
Error Access Denied occurred while performing operation %%2 on file %3
```

The next substitution replaces "%%2" with "Modify", resulting in:

```
Error Access Denied occurred while performing operation Modify on file %3
```

Finally, the last substitution replaces "%3" with "C:\securestuff\noaccess.db", resulting in the event description string:

```
Error Access Denied occurred while performing operation Modify on file
C:\securestuff\noaccess.db
```

5 Security

The following sections specify security considerations for implementers of the EventLog Remoting Protocol.

5.1 Security Considerations for Implementers

Implementers MUST take care to enforce the read/write permissions specified in section 3.1.4 to prevent unauthorized access to event logs.

Note Server prerequisites required for remote event description rendering (as defined in section 1.5.1) may make the server more at risk to security attacks, and therefore should be applied with caution.

5.2 Index of Security Parameters

Security parameter	Section
Authentication Service	2.1

6 Appendix A: Full IDL

```
import "ms-dtyp.idl";

[
    uuid(82273FDC-E32A-18C3-3F78-827929DC23EA),
    version(0.0),
    pointer default(unique)
]


interface eventlog
{
// the following line(s) commented out to avoid redefinition of MS-DTYP types
//typedef long NTSTATUS;

#define MAX_STRINGS      0x00000100
#define MAX SINGLE EVENT 0x0003FFFF
#define MAX BATCH BUFF   0x0007FFFF


typedef struct _RPC_STRING
{
    unsigned short Length;
    unsigned short MaximumLength;
    [size is(MaximumLength)] char* Buffer;
} RPC_STRING,  *PRPC_STRING;

typedef struct  RPC CLIENT ID {
    unsigned long UniqueProcess;
    unsigned long UniqueThread;
} RPC CLIENT ID, *PRPC CLIENT ID;

typedef [handle, unique] wchar_t * EVENTLOG_HANDLE_W;
typedef [handle, unique] char *  EVENTLOG_HANDLE_A;
typedef [context handle] void * IELF HANDLE;
typedef [context handle] void ** PIELF HANDLE;
typedef [range(0, MAX BATCH BUFF)] unsigned long RULONG;


NTSTATUS
ElfrClearELFW (
    [in]           IELF HANDLE LogHandle,
    [in,unique]    PRPC UNICODE STRING BackupFileName
    );

NTSTATUS
ElfrBackupELFW (
    [in]         IELF HANDLE LogHandle,
    [in]         PRPC UNICODE STRING BackupFileName
    );

NTSTATUS
ElfrCloseEL (
    [in,out]        IELF HANDLE * LogHandle
    );

NTSTATUS
ElfrDeregisterEventSource (
    [in,out]        IELF HANDLE * LogHandle
    );

NTSTATUS
```

```
ElfrNumberOfRecords(
    [in]            IELF_HANDLE         LogHandle,
    [out]           unsigned long *     NumberOfRecords
    );

NTSTATUS
ElfrOldestRecord(
    [in]            IELF_HANDLE         LogHandle,
    [out]           unsigned long *     OldestRecordNumber
    );

NTSTATUS
ElfrChangeNotify(
  [in] IELF_HANDLE* LogHandle,
  [in] RPC_CLIENT_ID ClientId,
  [in] unsigned long Event
);

NTSTATUS
ElfrOpenELW (
    [in]    EVENTLOG HANDLE W UNCServerName,
    [in]           PRPC UNICODE STRING ModuleName,
    [in]           PRPC UNICODE STRING RegModuleName,
    [in]           unsigned long MajorVersion,
    [in]           unsigned long MinorVersion,
    [out]          IELF_HANDLE * LogHandle
    );

NTSTATUS
ElfrRegisterEventSourceW (
    [in]          EVENTLOG HANDLE W UNCServerName,
    [in]          PRPC_UNICODE_STRING ModuleName,
    [in]          PRPC_UNICODE_STRING RegModuleName,
    [in]          unsigned long MajorVersion,
    [in]          unsigned long MinorVersion,
    [out]         IELF HANDLE * LogHandle
    );

NTSTATUS
ElfrOpenBELW (
    [in]          EVENTLOG HANDLE W UNCServerName,
    [in]          PRPC UNICODE STRING BackupFileName,
    [in]          unsigned long MajorVersion,
    [in]          unsigned long MinorVersion,
    [out]         IELF HANDLE * LogHandle
    );

NTSTATUS
ElfrReadELW (
    [in]         IELF_HANDLE     LogHandle,
    [in]         unsigned long ReadFlags,
    [in]         unsigned long RecordOffset,
    [in]         RULONG NumberOfBytesToRead,
    [out, size is(NumberOfBytesToRead)] unsigned char * Buffer,
    [out]        unsigned long * NumberOfBytesRead,
    [out]        unsigned long * MinNumberOfBytesNeeded
    );

NTSTATUS
ElfrReportEventW (
    [in]         IELF HANDLE LogHandle,
    [in]         unsigned long Time,
    [in]         unsigned short EventType,
    [in]         unsigned short EventCategory,
    [in]         unsigned long EventID,
```

```
    [in, range(0, 256)]       unsigned short NumStrings,
    [in, range(0, 61440)]  unsigned long DataSize,
    [in]         PRPC_UNICODE_STRING ComputerName,
    [in, unique] PRPC SID UserSID,
    [in, size is(NumStrings), unique] PRPC UNICODE STRING Strings[*],
    [in, size is(DataSize), unique] unsigned char * Data,
    [in]         unsigned short Flags,
    [in,out,unique] unsigned long * RecordNumber,
    [in,out,unique] unsigned long * TimeWritten
    );

NTSTATUS
ElfrClearELFA (
    [in]         IELF_HANDLE LogHandle,
    [in,unique]  PRPC_STRING BackupFileName
    );

NTSTATUS
ElfrBackupELFA (
    [in]         IELF_HANDLE LogHandle,
    [in]         PRPC STRING BackupFileName
    );

NTSTATUS
ElfrOpenELA (
    [in]     EVENTLOG_HANDLE_A UNCServerName,
    [in]     PRPC_STRING ModuleName,
    [in]     PRPC STRING RegModuleName,
    [in]     unsigned long MajorVersion,
    [in]     unsigned long MinorVersion,
    [out]    IELF HANDLE * LogHandle
    );

NTSTATUS
ElfrRegisterEventSourceA (
    [in]     EVENTLOG HANDLE A UNCServerName,
    [in]     PRPC STRING ModuleName,
    [in]     PRPC_STRING RegModuleName,
    [in]     unsigned long MajorVersion,
    [in]     unsigned long MinorVersion,
    [out]    IELF HANDLE * LogHandle
    );

NTSTATUS
ElfrOpenBELA (
    [in]     EVENTLOG HANDLE A UNCServerName,
    [in]     PRPC STRING FileName,
    [in]     unsigned long MajorVersion,
    [in]     unsigned long MinorVersion,
    [out]    IELF_HANDLE * LogHandle
    );

NTSTATUS
ElfrReadELA (
    [in]    IELF_HANDLE LogHandle,
    [in]    unsigned long ReadFlags,
    [in]    unsigned long RecordOffset,
    [in]    RULONG NumberOfBytesToRead,
    [out, size is(NumberOfBytesToRead)] unsigned char * Buffer,
    [out]   unsigned long * NumberOfBytesRead,
    [out]   unsigned long * MinNumberOfBytesNeeded
    );

NTSTATUS
ElfrReportEventA (
```

```
    [in]    IELF HANDLE LogHandle,
    [in]    unsigned long Time,
    [in]    unsigned short EventType,
    [in]    unsigned short EventCategory,
    [in]    unsigned long EventID,
    [in, range(0, 256)]    unsigned short NumStrings,
    [in, range(0, 61440)]    unsigned long DataSize,
    [in]    PRPC_STRING ComputerName,
    [in, unique] PRPC_SID UserSID,
    [in, size is(NumStrings), unique] PRPC STRING Strings[*],
    [in, size is(DataSize), unique] unsigned char * Data,
    [in]    unsigned short Flags,
    [in,out,unique] unsigned long * RecordNumber,
    [in,out,unique] unsigned long * TimeWritten
    );

void Opnum19NotUsedOnWire(void);
void Opnum20NotUsedOnWire(void);
void Opnum21NotUsedOnWire(void);

NTSTATUS
ElfrGetLogInformation(
    [in]     IELF HANDLE             LogHandle,
    [in]     unsigned long                   InfoLevel,
    [out, size_is(cbBufSize)] unsigned char *  lpBuffer,
    [in, range(0, 1024)]      unsigned long  cbBufSize,
    [out]    unsigned long *                  pcbBytesNeeded
);

void Opnum23NotUsedOnWire(void);

NTSTATUS
ElfrReportEventAndSourceW (
    [in]         IELF HANDLE LogHandle,
    [in]         unsigned long Time,
    [in]         unsigned short EventType,
    [in]         unsigned short EventCategory,
    [in]         unsigned long EventID,
    [in]         PRPC_UNICODE_STRING SourceName,
    [in, range(0, 256)]         unsigned short NumStrings,
    [in, range(0, 61440)]         unsigned long DataSize,
    [in]         PRPC UNICODE STRING ComputerName,
    [in, unique] PRPC_SID UserSID,
    [in, size_is(NumStrings), unique] PRPC_UNICODE_STRING Strings[*],
    [in, size is(DataSize), unique] unsigned char * Data,
    [in]         unsigned short Flags,
    [in,out,unique] unsigned long * RecordNumber,
    [in,out,unique] unsigned long * TimeWritten
    );

}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2003
- Windows Server 2008
- Windows NT Workstation 4.0 SP2 and later
- Windows 2000
- Windows XP
- Windows Vista

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.4: The EventLog Remoting Protocol is often used in conjunction with the Windows Remote Registry Protocol, as specified in [MS-RRP]. This is because several aspects of the event log are not configured through the EventLog Remoting Protocol; rather, they are configured by modifying the registry.

The EventLog Remoting Protocol Version 6.0 is a replacement for this protocol and is available on Windows Vista.

<2> Section 1.6: On Windows Vista, the EventLog Remoting Protocol Version 6.0 is preferred because of its additional functionality.

The EventLog Remoting Protocol is supported on Windows NT Workstation 4.0 SP2 and later, Windows 2000, Windows XP, Windows Server 2003, and Windows Vista. Note that the Windows client platforms such as Windows XP and Windows Vista may act as either a client or a server for the purpose of this protocol. Similarly, the Windows server platforms such as Windows Server 2003 and Windows Server 2008 can also act as either a client or a server for the purpose of this protocol.

<3> Section 1.8.1: Windows only uses the values in [MS-DTYP].

<4> Section 1.8.2: Windows does not prefix the names of the event logs it creates. In addition, Windows implementations impose the following limitations on event log names: They MUST be treated in a case-insensitive manner, they MUST be limited to 200 characters (400 bytes if Unicode is used), and they MUST NOT begin with the character '\'.

<5> Section 1.8.3: Windows does not prefix the names of the event sources it creates. In addition, Windows implementations impose the following limitations on event source names: They MUST be treated in a case-insensitive manner, they MUST be limited to 200 characters (400 bytes if Unicode is used), and they MUST NOT begin with the character '\'.

<6> Section 2.2.4: The event sources that write to the Windows security log use the EVENTLOG_AUDIT_SUCCESS and EVENTLOG_AUDIT_FAILURE types exclusively, whereas event sources that write to other logs use the other four types exclusively.

<7> Section 2.2.5: Windows sends NetBIOS names, as specified in [MS-SMB].

<8> Section 2.2.5: 32-bit Windows machines use 0 bytes of padding. 64-bit Windows machines use a number of bytes of padding needed to make the end of this field be on an 8-byte boundary from the beginning of the structure.

<9> Section 2.2.12: One of the most common code pages is ANSI Latin-1 (as specified in [ISO-8859-1]), and so the term ANSI is used.

<10> Section 2.2.12: For information on how Windows converts between Unicode and ANSI strings, see [MSDN-ANSI] and [MSDN-TRANS].

<11> Section 3: The NDR consistency check is at target level 5.0 (Windows versions earlier than Windows Vista) or target level 6.0 (Windows Vista), as specified in [MS-RPCE] section 3.

<12> Section 3.1.1.2: The log subkey also specifies log attributes such as its maximum size and its retention settings. The retention settings determine how the server handles events after the log reaches its maximum size. The retention can be set to either fail all new writes, or to start overwriting the oldest records. In the latter case, the log is treated as a circular log. The Eventlog Remote Protocol does not support any RPC methods for getting or setting the maximum event log size or its retention policy. For more information, see [MSDN-EVENTS].

<13> Section 3.1.1.3: If a client attempts to register an event source that does not exist in the registry under any of the event logs, the event log service still allows the client to succeed, and writes these events to the application event log, creating this log if it did not already exist.

<14> Section 3.1.1.4: In Windows, the EventID is mapped to a description related to the event by using a separate file, where the file is specific to the event source. As specified in section 3.1.1.3, it is possible to write events under event sources that do not exist in the registry. If the EventID is relative to an event source that does not exist in the registry, any clients that are reading events will not be able to find a description for any of the EventIDs.

The EventID layout is used by other operating system components besides the event log. Because of this, the layout used by Windows has some additional structure (for example, a Facility field and a Code field) that is not used by the event log and that can be ignored in this context.

<15> Section 3.1.3: By default, events produced by unregistered publishers are sent to the application event log. Therefore, the server creates a live event log with the name Application, if one does not already exist. If creation of the event log (with the name Application) fails, the EventLog Remoting Protocol server does not start.

<16> Section 3.1.4: All errors are as specified in [MS-ERREF] section 4. If **ElfrChangeNotify** is called remotely, Windows–based server implementations typically return STATUS_INVALID_HANDLE, as specified in [MS-ERREF].

<17> Section 3.1.4: The server has an access control list (ACL) that is used to control access to the log. The protocol has no methods for reading or setting that ACL.

<18> Section 3.1.4.1: In Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, failures other than checks on the *BackupFileName* parameter erroneously return STATUS_SUCCESS (0x00000000) with *LogHandle* set to NULL.

<19> Section 3.1.4.2: In Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008, failures other than checks on the *BackupFileName* parameter erroneously return STATUS_SUCCESS (0x00000000) with *LogHandle* set to NULL.

<20> Section 3.1.4.3: The server has an access control list (ACL) that is used to control access to the log. The protocol does not have any methods for reading or setting that ACL.

<21> Section 3.1.4.9: In Windows Vista and Windows Server 2008s, the methods do not differentiate between handles for event log files and handles for backup event log files. These methods return STATUS_SUCCESS when called with a handle obtained from **ElfrOpenBELA** (specified in section 3.1.4.2 or **ElfrOpenBELW** (specified in section 3.1.4.3).

<22> Section 3.1.4.13: The API is not intended to support dynamically changing computer names. Current implementations of Windows cache the *ComputerName* parameter the first time a publisher calls the API, and use that name on subsequent calls until the machine is rebooted.

<23> Section 3.1.4.13: In the Windows Vista and Windows Server 2008s, the server does not set these values. Thus, they retain the values set by the client.

<24> Section 3.1.4.14: The API is not intended to support dynamically changing computer names. Current implementations of Windows cache the *ComputerName* parameter the first time a publisher calls the API, and use that name on subsequent calls until the machine is rebooted.

<25> Section 3.1.4.15: This method is supported only on Windows Server 2003 R2.

<26> Section 3.1.4.21: Windows implementations typically as specified in [MS-ERREF] section 4.

<27> Section 3.1.4.21: Windows implementations typically return STATUS_INVALID_HANDLE or STATUS_INVALID_PARAMETER, as specified in [MS-ERREF] section 4.

<28> Section 3.2.4: The server has an access control list (ACL) that is used to control access to the log. The protocol does not have any methods for reading or setting that ACL.

<29> Section 3.2.4: The server has an access control list (ACL) that is used to control access to the log. The protocol does not have any methods for reading or setting that ACL.

<30> Section 3.2.4: This method is only supported on Windows Server 2003 R2, Windows Vista, and Windows Server 2008. If **ElfrChangeNotify** is called remotely, Windows–based server implementations typically return STATUS_INVALID_HANDLE, as specified in [MS-ERREF].

<31> Section 3.2.4.1.1: In Windows client implementations, these are not read for the Security log due to that subkey's highly restrictive permissions; in this case, the log name is a resource in the Event Viewer application.

<32> Section 3.2.4.1.1: Based on knowledge of client preferred locales, Windows client implementations may try to load a resource string from an alternate resource library location. Windows client implementations for Windows SKUs earlier than Windows Vista may append a numeric locale identifier (LCID) such as "409" to the file path and an ".mui" extension to the file name. Windows SKUs after Windows Vista insert a language name such as "en-us". In either case, Windows SKUs fall back to the original file path if the language-specific file is not found. For more information on locale identifiers and language names, see [MS-LCID].

<33> Section 3.2.4.1.2: In Windows client implementations, these are not read for the Security log due to that subkey's highly restrictive permissions; in this case, the log name is a resource in the Event Viewer application.

<34> Section 3.2.4.1.2: Based on knowledge of client preferred locales, Windows client implementations may try to load a resource string from an alternate resource library location. Windows client implementations for WindowsSKUs earlier than Windows Vista may append a numeric locale identifier (LCID) such as "409" to the file path and an ".mui" extension to the file name. Windows SKUs in Windows Vista insert a language name such as "en-us". In either case, Windows SKUs fall back to the original file path if the language specific file is not found. For more information about locale identifiers and language names, see [MS-LCID].

<35> Section 3.2.4.1.3: In Windows client implementations, these are not read for the Security log due to that subkey's highly restrictive permissions; in this case, the log name is a resource in the Event Viewer application.

<36> Section 3.2.4.1.3: In Windows client implementations, these are not read for the Security log due to that subkey's highly restrictive permissions; in this case, the log name is a resource in the Event Viewer application.

<37> Section 3.2.4.1.3: Based on knowledge of client preferred locales, Windows client implementations may try to load a resource string from an alternate resource library location. Windows client implementations for Windows SKUs prior to Windows Vista may append a numeric locale identifier (LCID) such as "409" to the file path and an ".mui" extension to the file name. Windows SKUs in Windows Vista insert a language name such as "en-us". In either case, Windows SKUs fall back to the original file path if the language specific file is not found. For more information on locale identifiers and language names, see 3.2.4.1.5.2.

<38> Section 3.2.4.1.4: In Windows client implementations, these are not read for the Security log due to that subkey's highly restrictive permissions; in this case, the log name is a resource in the Event Viewer application.

<39> Section 3.2.4.1.4: Based on knowledge of client preferred locales, Windows client implementations may try to load a resource string from an alternate resource library location. Windows client implementations for Windows SKUs earlier than Windows Vista may append a numeric locale identifier (LCID) such as "409" to the file path and an ".mui" extension to the file name. Windows SKUs in Windows Vista insert a language name such as "en-us". In either case, Windows SKUs fall back to the original file path if the language specific file is not found. For more information on locale identifiers and language names, see [MS-LCID].

<40> Section 3.2.4.1.5: The replacement behavior is not exactly recursive, although it is very similar to recursive behavior. Consider the unexpanded description string "%1%2" where the first EventLogRecord.String is "%". This becomes "%%2", which is a parameter insertion, and the second EventLogRecord.String is never retrieved.

<41> Section 3.2.4.1.5: The number of substitutions in Windows implementations is capped at 100.

<42> Section 3.2.4.1.5.3: Expanding SIDs: Starting with Windows 2000, Windows client implementations attempt to look up the name of the security principal for a properly formatted SID. The lookup is first attempted on the event source server, and, if that fails, it is attempted in the Global Catalog server for the forest to which the event source server belongs. For information on how to implement this lookup, see [MS-LSAD] and [MS-LSAT].

Expanding GUIDs: Starting with Windows 2000, Windows client implementations attempt to find the name of the Active Directory object with this GUID. First, the client implementations attempt to look this up as a well-known schema GUID (for example, Administrators). Then, the client implementations look for an object by this name on the domain controller (DC) in the same domain as the target computer. Finally, they look for an object by this name on the Global Catalog for the local domain. If the client implementations still have not succeeded, they leave the GUID string in the output as is. For information on implementing this lookup, see [RFC2251] and [MS-ADTS].

<43> Section 3.2.4.1.5.4: As a fallback, Windows Event Viewer for SKUs later than Windows XP tries to resolve "%systemroot%" and "%systemdrive%" environment variables by reading the local registry value "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion" when it fails to read the remote server registry for the same value.

8 Index

N

O

P

R

S

T

U

V

W