Service Availability™ Forum System Management Specification

Log Service SAI-AIS-LOG-A.01.01

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

Table of Contents Log Service

1 Document Introduction

1.1 Document Purpose

This document defines the Log Service of the Application Interface Specification (AIS) of the Service Availability^{TM} Forum (SA Forum). It is intended for use by implementors of the Application Interface Specification and by application developers who would use the Application Interface Specification to develop applications that must be highly available. The AIS is defined in the C programming language, and requires substantial knowledge of the C programming language.

Typically, the Service Availability^{TM} Forum Application Interface Specification will be used in conjunction with the Service Availability^{TM} Forum Hardware Interface Specification (HPI) and the Service Availability^{TM} Forum System Management Specification.

1.2 AIS Documents Organization

The Application Interface Specification is organized into several volumes. For a list of all Application Interface Specification documents, refer to the SA Forum Overview document [4].

1.3 History

SAI-AIS-LOG-A.01.01 is the first release of the Log Service specification.

1.4 References

The following documents contain information that is relevant to this specification.

[1] CCITT Recommendation X.735 | ISO/IEC 10164-5, Log Control Function

[2] Service Availability^{TM} Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.01.01

[3] Service Availability^{TM} Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.01.01

[4] Service Availability^{TM} Forum, Application Interface Specification, Overview, SAI-Overview-B.02.01

[5] Service Availability^{TM} Forum, Hardware Platform Interface, SAI-HPI-B.02.01

[6] CCITT Recommendation X.733 | ISO/IEC 10164-4, Alarm Reporting Function

[7] IETF RFC 3164, The BSD Syslog Protocol

1.5 How to Provide Feedback on the Specification

If you have a question or comment about this specification, you may submit feedback online by following the links provided for this purpose on the Service Availability™ Forum website (**http://www.saforum.org**).

You can also sign up to receive information updates on the Forum or the Specification.

1.6 How to Join the Service Availability™ Forum

The Promoter Members of the Forum require that all organizations wishing to participate in the Forum complete a membership application. Once completed, a representative of the Service Availability™ Forum will contact you to discuss your membership in the Forum. The Service Availability™ Forum Membership Application can be completed online by following the pertinent links provided on the Forum's website (**http://www.saforum.org**).

You can also submit information requests online. Information requests are generally responded to within three business days.

1.7 Additional Information

1.7.1 Member Companies

A list of the Service Availability™ Forum member companies can also be viewed online by using the links provided on the Forum's website (**http://www.saforum.org**).

1.7.2 Press Materials

The Service Availability™ Forum has available a variety of downloadable resource materials, including the Forum Press Kit, graphics, and press contact information. Visit this area often for the latest press releases from the Service Availability™ Forum and its member companies by following the pertinent links provided on the Forum's website (**http://www.saforum.org**).

2 Overview

This specification defines the Log Service within the Application Interface Specification (AIS).

2.1 Log Service

SA Forum specifications distinguish between log and trace services. This specification does not support trace services. The distinction can be characterized as follows:

Logging information is a high level cluster-significant, function-based (as opposed to implementation-particular) information suited primarily for network or system administrators, or automated tools to review current and historical logged information to trouble shoot issues such as mis-configurations, network disconnects and unavailable resources.

Tracing information, on the other hand, is low level product and implementation-particular information suited primarily for developers or field engineers, often engaged in debugging implementation specifics such as timing, algorithms and distributed applications. A SA Forum Trace Service is on the roadmap, but is not yet defined.

A SA Forum compliant ecosystem assumes the AIS Log Service, or some functionally equivalent service is available for use by applications as well as other AIS services.

Some SA Forum services, such as the Notification Service (abbreviated as NTF, see [2]), explicitly expect a log service, such as the SA Forum Log Service, to be available.

SA Forum Hardware Platform Interface (HPI)[5] logging is not integrated with the SA Forum Log Service in this version of the document. This is left for future study with the intent of integrating these two in a subsequent version of this document.

The following diagram identifies the principle abstractions of the SA Forum Log Service.

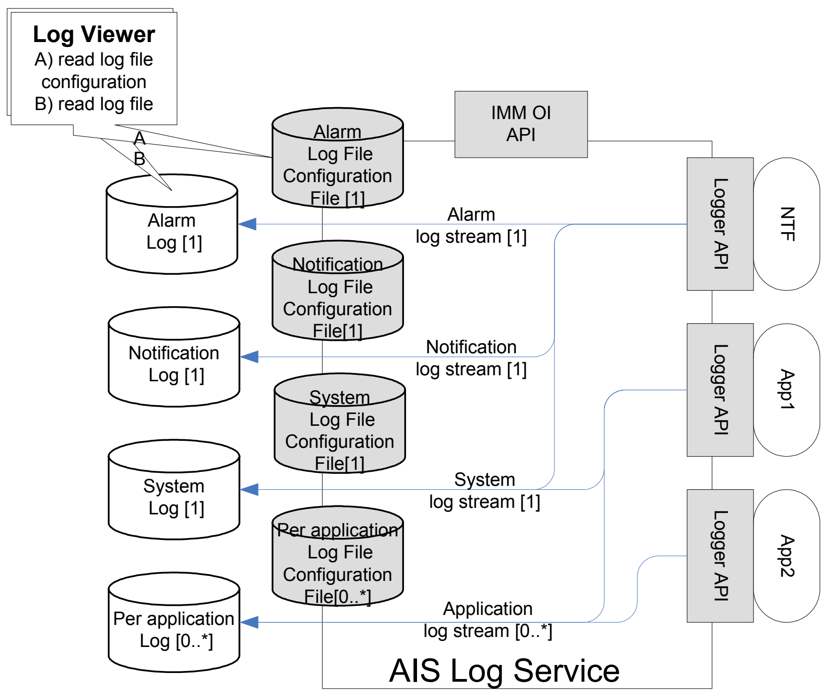
Log Viewer
A) read log file configuration
B) read log file
A
B
Alarm Log File Configuration File [1]
IMM OI API
Alarm Log [1]
Alarm log stream [1]
Logger API
NTF
Notification Log File Configuration File[1]
Notification Log [1]
Notification log stream [1]
System Log File Configuration File[1]
Logger API
App1
System Log [1]
System log stream [1]
Per application Log File Configuration File[0..*]
Logger API
App2
Per application Log [0..*]
Application log stream [0..*]
AIS Log Service

FIGURE 1 Log Service Entities

Within the SA Forum Log Service boundary, there are objects internal to the Log Service. They are:

- log stream - A log stream is a conceptual flow of log records. There are four distinct log stream types (alarm, notification, system, and application), which are explained in the next section 2.2 and then more extensively in section 3.1.2 .
- log record - A log record is an ordered set of information logged by some process (see section 3.1.3).

All grayed objects at the SA Forum Log Service boundary are public interfaces and are formally defined in this document. Briefly, these public interfaces are:

- Logger API - The logger API is a linkable library used by processes that wish to send a log record on a particular log stream (see section 3.5).
- Log File Configuration File - At an output destination of a particular log stream, there is a publicly readable 'log file configuration file' (see section 3.1.6.2) which explains the log file (or files) properties associated with that log stream, such as how the log record data is formatted for the associated log file or files (see section 3.1.5).

- IMM Object Implementer API - This is the Information Management Model (IMM) Service [3] Object Manager interface. It is not intended for consumers of the Log Service. Rather, it provides access to the Log Service objects as well as administrative operations associated with those objects. Clients of this interface would typically be system management applications such as SNMP agents or CIM providers.

The diagram also shows a 3rd party 'Log Viewer' that (A) first reads the log file configuration file which allows the viewer to (B) read and understand how the log records are formatted in the associated log file or files (see section 3.3.6.1). Such 'viewer' or 'reader' functionality is outside the scope of the SA Forum Log Service.

2.2 Log Streams

The Log Service enables applications to express and forward log records through well-known log streams that lead to particular output destinations such as a named file. A log record format expression explains how the fields of each log record shall be displayed at an output destination.

There are four types of log streams supported by the Log Service:

- The alarm log stream is for ITU X.733 and ITU X.736 based log records.
- The notification log stream is for ITU X.730 and ITU X.731 based log records.
- The system log stream is for system relevant log records.
- Application log streams are for application-specific log records.

There is exactly one log stream for each of the alarm, notification, and system log stream types in an SA Forum cluster. However, there can be any number of application log streams. The SA Forum Notification Service (NTF)[2] is envisioned as the principal user of the alarm and notification log streams, though other users are possible.

The SA Forum Log Service may define new log streams or augment existing streams with new log record types in some future revision of this specification.

2.3 Log Stream Handlers

The SA Forum Log Service also has the concept of log stream handlers, which is not specified in this release of the document but will be specified in a future release.

Roughly, a log stream handler will allow an administrator to copy or redirect 'matched' log records traveling through a particular log stream to a distinct output destination such as a log file, terminal or another program. Matched log records will then be subject to a log record format expression that is associated with that log stream handler.

Administrators will be able to configure any number of log stream handlers to a log stream.

3 SA Log Service API

3.1 Log Service Model

3.1.1 Logger

A **logger** is a client of the Log Service that uses the *saLogWriteLog()* API in order to introduce a **log record** to a specific **log stream**. A logger gains access to a log stream by invoking *saLogStreamOpen()* and can terminate its relationship with a log stream by invoking *saLogStreamClose()*.

3.1.2 Log Stream

A **log stream** is a conceptual flow of log records. Each log stream has a name that is unique in the cluster. Each log stream leads to an **output destination** log file or files (see Section 3.3.6.1). There are four distinct types of log streams supported by the Log Service:

1. **Alarm** log stream: The SA Forum Notification (NTF) Service [2] is presumed a client of this Log Service though it is not required. NTF logs alarm information as per the ITU documents alarm reporting (X.733) and security alarm reporting (X.736). Within a cluster, there is a single, well-known alarm log stream named '*safLgStr = saLogAlarm*', which leads to an output destination file that only contains these alarm log records.
2. **Notification** log stream: The SA Forum Notification (NTF) Service [2] is presumed a client of this Log Service though it is not required. NTF optionally logs notification information as per ITU documents object management (X.730) and state management (X.731). Within a cluster, there is a single, well-known notification log stream named '*safLgStr = saLogNotification*', which leads to an output destination file that only contains these notification log records.
3. **System** log stream: The system log stream is used by applications to record relevant and noteworthy system circumstances, particularly those that effect service. This log can also be used by AIS services as well as AMF to log cluster wide significant events. The data on this stream is less formal than alarm or notification log streams. Within a cluster, there is a single, well known system log stream named '*safLgStr = saLogSystem*', which leads to an output destination file that only contains these system log records.
4. **Application** log stream: An application log stream can be created and used by an application that wants certain log records isolated from the system log. Each application can create its own application log stream or open an existing application log stream using *saLogStreamOpen()*. There can be any number of application log streams in a cluster at one time, and they can dynamically come and go.

Log records on one stream do not mingle with log records on any of the other log streams.

The transport requirements for these log streams are guaranteed and in-order delivery from any given logger source to its final output destination.

3.1.2.1 Alarm, Notification, and System Log Streams

The alarm, notification and system log streams are distinct, well-known cluster-wide log streams that can neither be created or destroyed. Each of these three log streams leads to a stream specific, mandatory system defined log file or files (see Section 3.3.6.1) that also has an associated log file configuration file (see Section 3.1.6.2).

Log file configuration file attributes can be configured through administrative means very early in the life of the cluster through the IMM interface [3]. If no configuration is provided, an implementation-specific default configuration shall be applied to these log streams.

The alarm, notification and system log streams are made active when the Log Service successfully initializes and is available for service.

3.1.2.2 Application Log Stream

Application loggers can create private application log streams at runtime by way of the *saLogStreamOpen()* API. The application logger must specify both a file (Section 3.1.6) and format (Section 3.1.5) configuration. This configuration applies to all log records placed on that log stream by way of *saLogWriteLog()*. Any number of application loggers can join an existing application log stream using the *saLogStreamOpen()* API by identifying the same log stream by it *logStreamName* and either:

- Specifying no other create properties (since the log stream and its properties already exist), or
- Specifying exactly the same create properties of the already existing log stream. If create properties are specified, but do not match, it is an error.

There can be any number of private application log streams in a cluster at any given time, though each must go by a cluster wide unique name. The same application can also call *saLogStreamOpen()* to open more than one application log stream at the same time.

An application log stream is destroyed when all application loggers using that stream *saLogStreamClose()* it. The output destination log file or files (see Section 3.3.6.1) and log file configuration file (see Section 3.1.6.2) associated with the destroyed log stream is closed and persists indefinitely.

If another application log stream is created using *saLogStreamOpen()* with the same *logStreamName* and *saLogFilePathName* as a previously destroyed log stream and other *saLogStreamCreateAttributeT values* are either the same or different, the Log Service (and log readers) can distinguish this new log stream from any predecessors by inspection of the log file name changes that have been automatically applied by the Log Service to all completed log files (see Sections 3.1.6.2 and 3.1.6.3).

3.1.3 Log Record Properties

Log records travel through a log stream toward an output destination. The Log Service is not required to interleave log records on a log stream based on log record's *logTimeStamp* (time stamp). Rather, log records can be interleaved on a log stream on a first-to-arrive basis.

In fact, the Log Service makes no internal decisions based on *logTimeStamp* values. The Log Service places no firm requirements regarding clock synchronization in a distributed system.

3.1.4 Log Filtering

Log filtering applies to application and system log records only.

Log filtering means that only matched log records are allowed entry onto a log stream; all others are discarded. A log filter criteria can only be accessed and configured through administrative means.

Log filtering of alarm or notification log records is not supported since the SA Forum log philosophy is that all published alarms and notifications must be logged. Notice, the SA Forum Notification Service [2] has a concept of non-alarm filtering, but this would happen prior to and outside the scope of Log Service awareness.

A log filter criteria is based on:

- The severity value of a system or application log record

Other filter criteria can be imagined and may be introduced in future revisions of this document. For example, a filter criteria may qualify that particular nodes, applications or service units shall be allowed to log. Such imagined criteria would be considered in conjunction with the existing severity filter criteria.

Log filtering behavior is experienced by a logger as follows:

- The *(*saLogFilterSetCallbackT)()* callback explains to a logger its current filter criteria. This allows a logger to avoid the overhead of packaging and invoking the *saLogWriteLog()* for those log records that the Log Service will discard anyway.
- The Log Service itself also reviews introduced log records against the current filter criteria and discards any that do not match. This is done regardless of

whether a logger provided a *(*saLogFilterSetCallbackT)()* function pointer at *saLogInitialize()* time or not.

3.1.5 Log Record Output Format

Log record output **formatting rules** consist of a well-known set of log record **format tokens** that can be ordered into well formed log record **format expressions,** which governs the output properties of each log record at an output destination.

Each format token maps to a specific field or sub-field in a log record. A format token also implies a specific output display. A format expression is a sequence of these format tokens, which as a whole, explains the presence, order and format of how log record fields are to be displayed.

Log record format expression rules must be formally described since such expressions serve as a public interface of the Log Service. Precise syntax ensures that third party tools can read and manipulate Log Service output such as log files since such log file ‘reader’ tools are outside the scope of this Log Service.

The Log Service provides a means to configure a format expression at each output destination. A default format expression is applied if no format expression is configured or a configured format expression is illegal (not well formed). Once an output destination is made operational, the associated format expression cannot change for the life of that output destination. This guarantees that all log records delivered to a particular output destination are formatted the same way.

3.1.5.1 Format Tokens

There is a set of simple format tokens that are used to both identify fields or subfields of a log record and to express the desired output form of that field.

Each token type either implicitly or explicitly identifies the number of character spaces associated with that token’s output. The cumulative effect is that each field in a log record can be placed at fixed offsets so that all output records at the same output destination are formatted identically. This allows a log reader to easily calculate offsets into specific log records within a log file.

The formal representation of a format token is:

```
<@><C|S|N><letter><field-size>
```

which breaks down to these parts:

<@ >All token sequences start with the ‘at’ symbol

<C|S|N> the next character indicates if it is:

- C = A common log record field, or

- S = A system or application log record field, or
- N = A notification or alarm field

<letter> a distinct character that maps to a specific field or subfield of a log record.

<field-size> most token types imply a fixed output field size and cannot be followed by this field size qualifier. However, some token types optionally allow its output field size to be specified.

- If allowed and specified by the user, the output will occupy exactly <field-size> spaces either by adding blanks or truncating a long string.
- If not specified but allowed, then the output will use exactly the number of spaces it takes to express the value. This results in variable field offsets from log record to log record at the same output destination.

An example token is:

`@Sl30`

This is a system or application (S) token for the *logSvcUsrName* field (the letter '`l`'). It will occupy exactly 30 spaces.

The table below shows the complete set of format tokens available for constructing format expressions. These tokens track to specific fields or subfields of the *SaLogRecordT* data type(see Section 3.3.5.5).

- The left column shows each token type syntax supported by the Log Service. The token types that end with <fs> can optionally be configured with a numeric <field-size> value.
- The center column describes format rules and semantics.
- The right column is an arbitrary example of legal output ('.' is used here to make clear the number spaces that would otherwise appear as blanks. The '.' is not a Log Service output requirement).

Table 1: Log Record Format Tokens

Token Type	Description	Example Output Format
@Cr	A 10 digit log record Identifier that the Log Service internally generates. This unsigned 32bit numeric assignment starts at 1 and increments by 1 as log records arrive at the particular output destination (see Section 3.1.6.4).	`'.......345'`

Table 1: Log Record Format Tokens

Token Type	Description	Example Output Format
@Ct	18 character hexadecimal representation of time from *logTimeStamp* of type *saTimeT* in the *SaLogRecordT* structure (see Section 3.3.5.5). This time is when a log record was actually logged.	`0x0006670634553455`
@Ch	2 digit hour of the day from *logTimeStamp* of type *saTimeT.* If the common token type @Ca (for am/pm output) is in a format expression, then the output is formatted for a 12 hour clock. Otherwise the output is formatted for a 24 hour clock.	`04`
@Cn	2 digit minute of the hour from *logTimeStamp* of type *saTimeT.*	`45`
@Cs	2 digit second of the minute from*logTimeStamp* of type *saTimeT.*	`08`
@Ca	am/pm according to a 12 hour clock, from *logTimeStamp* of type *saTimeT.* See token type @Ch.	`am`
@Cm	2 digit month from *logTimeStamp* of type *saTimeT.*	`10`
@CM	3 letter abbreviation for month from *logTimeStamp* of type *saTimeT.*	`Oct`
@Cd	3 letter day of the week from *logTimeStamp* of type *saTimeT.*	`Mon`
@Cy	2 digit year from *logTimeStamp* of type *saTimeT.*	`05`
@CY	4 digit year from *logTimeStamp* of type *saTimeT.*	`2005`

Table 1: Log Record Format Tokens

Token Type	Description	Example Output Format
@Cc	29 spaced Notification class identifier from *notificationClassId* of type *saNtfClassIdT [2].* The *vendorid*, *majorId* and *minorId* values are expressed as hexadecimal. Notice that the ‘NCI’ prefix, brackets and commas are implicit features of this output formatting.	`NCI[0x000346f1,0x0034,0x012a]`
@Cx	a single character that indicates if this log record’s output has been truncated to remain within its configured fixed log record size (see Section 3.1.6.2). The output values are: • ‘T’ means truncated • ‘C’ means complete	`T`
@Cb<fs>	If this token is used, the body of the log record from *logBuffer* of type *saLogBufferT* is assumed a printable string (see Section 3.3.2.3). If a \0 is found prior to the <fs> length, then blank characters will be applied for the remaining characters up to <fs>.	`“port access denied..” where <fs>=20`
@Ci<fs>	If this token is used, the body of the log record from *logBuffer* of type *saLogBufferT* is output as hexadecimal characters (see Section 3.3.2.3). If the *logBufSize* is less then <fs>, then blank characters will be applied for the remaining characters up to <fs>.	`“706f72742061636365737320646e6965642020”, where <fs>=40; (ascii= “port access denied ”)`
@Sl<fs>	logger name from *logSvcUsrName* of type *saNameT* (see Section 3.3.5.3).	`‘safSu=xx,safSg=yy,safApp=zz...’, where <fs>=30`

Table 1: Log Record Format Tokens

Token Type	Description	Example Output Format
@Sv	2 character severity identifier that maps to one of the *SA_LOG_SEV_* severity values (see Section 3.3.2.2). The identifiers are: • EM for *EMERGENCY* • AL for *ALARM* • CR for *CRITICAL* • ER for *ERROR* • WA for *WARNING* • NO for *NOTIFICATION* • IN for *INFO*	`CR`
@Ni	18 character hexadecimal representation of Notification id of type *saNtfIdentifierT [2],* a field in the *SaLogNtfLogHeaderT* structure (see Section 3.3.5.2)	`0x0000000000000043`
@Nt	18 character hexadecimal representation of time from *eventTime* of type *saTimeT,* a field in the *SaLogNtfLogHeaderT* structure (see Section 3.3.5.2). Notice that this time is when an alarm or notifications occurred, which is distinct from when a log record is logged (see @Ct).	`0x0006670634553455`
@Nh	2 digit hour of the day from *eventTime* of type *saTimeT.* If the common token type @Na (for am/pm output) is in a format expression, then the output is formatted for a 12 hour clock. Otherwise the output is formatted for a 24 hour clock.	`04`
@Nn	2 digit minute of the hour from *eventTime* of type *saTimeT.*	`05`
@Ns	2 digit second of the minute from *eventTime* of type *saTimeT.*	`47`

Table 1: Log Record Format Tokens

Token Type	Description	Example Output Format
@Na	am/pm according to a 12 hour clock, from *eventTime* of type *saTimeT.* See token type @Nh.	`pm`
@Nm	2 digit month from *eventTime* of type *saTimeT.*	`04`
@NM	3 letter abbreviation for month from *event-Time* of type *saTimeT.*	`Jan`
@Nd	3 letter day of the week from *eventTime* of type *saTimeT.*	`Fri`
@Ny	2 digit year from *eventTime* of type *saTimeT.*	`11`
@NY	4 digit year from *eventTime* of type *saTimeT.*	`2011`
@Ne<fs>	<field-size> hexadecimal expression for event type from type *saNtfEventTypeT [2],* a field in the *SaLogNtfLogHeaderT* structure (see Section 3.3.5.2)*. The* hex expression makes it easier for a human reader to identify the previously ORed parts of it.	`‘0x3002’, where <fs>=6 (which corresponds to SA_NTF_ATTRIBUTE_REMOVED).`
@Na<fs>	*notificationObject* of type *saNameT,* a field in the *SaLogNtfLogHeaderT* structure (see Sec-tion 3.3.5.2)	`‘safSu=xx,safSg=yy,safApp=zz........’, where <fs>=35`
@Ng<fs>	*notifyingObject* of type *saNameT,* a field in the *SaLogNtfLogHeaderT* structure (see Sec-tion 3.3.5.2)	`‘safSu=xx,safSg=yy,safApp=zz........’, where <fs>=35`

There are distinct but parallel time-related tokens for both common (C) and alarm and notification (N) record fields since the time when an alarm or notification is published and the time when that alarm or notification is logged are different times.

Also notice that all output is printable text, so that some amount of human inspection of log record output is possible without the aid of a log reader program.

3.1.5.2 Format Expressions

These format token types are sequenced to form log record format expressions that are subject to these rules.

1. It is an error to use a particular token type in a format expression that is incompatible with the log stream that the expression is associated with. This means:
   - Only @C and @S tokens can be used in a format expression that is associated with an application or system log stream.
   - Only @C and @N tokens can be used in a format expression that is associated with a notification or alarm log stream.
2. If a <field-size> is allowed and expressed for a particular token type, then

- All character output is left justified within its <field-size>. If the output is too big, the tail of the character output is truncated.
- All digit or hex output is right justified within its <field-size>. If the output is too big, the most significant digits or hex positions are truncated.

3. It is an error to reference the same token type more than once per format expression.
4. Literal characters placed in a format expression are output as is, in place (see Section 3.1.5.3). The exception is the @ character, which is reserved. It cannot be used as a literal. No escape sequence is defined.
5. For token types that format the identified field to a printable string (such as @Cb), any non-printable characters are output as under-bar (“_”). Some other substitute character may be defined as an implementation option.

The Log Service shall also place termination character(s) at the final character position(s) of each output log record. The actual character or characters used are implementation-specific, but the intent is to match ‘carriage return line feed’ semantics (different operating systems have their preferences). These characters are included in the fixed size total of each log record (see Section 3.1.6.2).

3.1.5.3 Default Format Expressions

If a log record format expression is not explicitly configured at an output destination, then the Log Service will us a default format expression.

The default log record format expression for the application and system log streams are:

```
@Cr @Ch:@Cn:@Cs @Cm/@Cd/@CY @Sv @Sl “@Cb”
```

This produces a formatted output like:

```
        ........33 04:35:45 05/22/2005 3 safSu=xx,safSg=yy,safApp=zz
"port access denied"
```

Notice in the example that the literal characters `[:, ,/,"]` placed in the format expression appear in the formatted output in the corresponding places. Also notice that the token types for *logSvcUsrName* (@Sl) and *logBuffer* (@Cb) fields are not qualified by a <field-size> value, so the field sizes for those tokens will be different for each log record in the log file.

The default log record format expression for the notification and alarm log streams are:

```
@Cr @Ct @Nt @Ne5 @Na30 @Ng30 "@Cb"
```

This produces a formatted output like:

```
        ...4563419 0x0006670634553455 0x0006670634553455 ...76
safSu=xx,safSg=yy,safApp=zz...safSu=xx,safSg=yy,safApp=zz... "port
access denied"
```

3.1.6 Log File Properties

The alarm, notification and system log streams each lead to their respective output files, where either a supplied log file configuration or a default log file configuration is applied. For these three cases, a configuration can be supplied through the IMM[3] service interface available very early in the life of the Log Service. If a log file configuration is not supplied, the Log Service shall use a default configuration. If a file configuration is supplied but has errors, the Log Service shall use a default configuration.

The actual values of a default configuration are implementation-specific as long as the default profile is legal, as outlined in Section 3.1.6.2.

For an application log stream however log file properties are configured by the logger when it creates a new application log stream when *saLogOpenStream()* is invoked. In this case, the configuration supplied must be correct in order for the stream to be created (see Section 3.5.1). There is no concept of a default set of log file properties.

From an external point of view, log stream log file properties can be learned in one of two ways:

- by way of IMM[3], where current application log stream properties are identified in runtime and configuration objects, or
- by subscribing to the 'log stream created' object change notification (see Section 5.2.2), which contains the data points necessary to know the name and location of the <filename>.cfg file (see Section 3.1.6.2), which explains the pertinent configuration information necessary to 'read' the corresponding log file.

Once an output destination is made operational, the associated file configuration cannot change for the life of that output destination.

3.1.6.1 Log File Configurable Attributes

The log file configurable attributes are:

log file path: this standard POSIX path name explains which directory the log file (or files) shall be placed. Details regarding where log files can live and how location within a cluster is specified is implementation-specific.

log file name: this name is used to create (at least) two files.

- <filename>.cfg, which contains the format expression and key configuration information associated with the log output files, and
- <filename>_<createtime>.log, which houses the logging information so formatted starting at <createtime> time.

maximum log file size: The maximum size a log file may grow, in bytes. Zero means there is no predefined limit.

fixed log record size: indicates the fixed log record size (after the formatting rules have been applied) that can be written to this file. Log record output smaller than this size are padded with blank characters. Log record output larger than this is truncated at the fixed log record size. This size includes Log Service termination characters, as described in Section 3.1.5.2.

high availability flag: Indicates if the log file must always be available and implies file replication and persistency. The implementation can achieve replication in any fashion it desires (replication, RAID storage, NAS/SAN, etc.) so long as it is accessible from the same path name from any node in the cluster. Persistency means that the log file must exist across cluster reboots (i.e., all nodes go down, then come back, thus for some period of time there is no cluster). High availability is always TRUE for the alarm and notification log files.

log file full action: explains the desired Log Service behavior when a file’s maximum log size is reached. The options are:

- wrap – Once the maximum log file size has been reached, the oldest log records are deleted as needed to allow for new log records to be added.
- halt – The log is full. No more log records are allowed in this file. For this action, ‘capacity alarm condition’ attributes may be configured, though such a configuration is left as an implementation matter in this release.
  - A ‘capacity alarm’ notification (see Section 5.2.1.2) shall be generated by the Log Service when capacity alarm conditions are reached.

- rotation – When the current log file is full, a new log file is created (with cre- atetime>) to which future log records are now written. For this action, these other attributes must also be configured.
    - max number of files: the maximum number of files allowed in the rotation. If the maximum number is reached, then the oldest file is removed and another file is then created.

3.1.6.2 Log File Configuration File

When an output destination is configured with a log <filename>, several files are created, and certain naming conventions are expected.

<filename>.cfg - The Log Service creates this log file configuration file prior to the log stream becoming operational. This file explains these key log file properties:

- The version of the Log Service that generated this file
- The log record format expression applied to the output (see Section 3.1.5.2).
- The maximum log file size configured
- The fixed size of each log record in the file
- Log file full action

The syntax of how these values appear in the <filename>.cfg must be formally described as it is a public interface of the Log Service. This specification allows any SA Forum standards based log file reader to parse the content and understand how to read the corresponding log files. The following BNF explains this syntax:

```
LogFileCfg          :    <LogVerExp> <FmatExp> <CfgExp>
LogVerExp           :    LOG_SVC_VERSION: <Version>
FmatExp             :    FORMAT: <LogRecFmatExp>
CfgExp              :    MAX_FILE_SIZE: <number>
                         FIXED_LOG_REC_SIZE: <number>
                         LOG_FULL_ACTION: <Action>
Action              :    WRAP
                    |    HALT
                    |    ROTATE <NumFilesToRotate>
Version             :    <ReleaseCode>.<MajorVers>.<MinorVers>
ReleaseCode         :    <character>
MajorVers           :    <number>
MinorVers           :    <number>
NumFilesToRotate    :    <number>
LogRecFmatExp       :    <see Section 3.1.5>
number              :    [0..9]+
```

An example of a legal <filename>.cfg file is:

```
LOG_SVC_VERSION: B.2.1
FORMAT:@Cr @Ch:@Cn:@Cs @Cm/@Cd/@CY @Sv @Sl "@Cb"
MAX_FILE_SIZE: 8000000
FIXED_LOG_REC_SIZE: 100
LOG_FULL_ACTION: ROTATE 4
```

This particular example <filename>.cfg file uses the default system an application log record format expression.

When the log file or files associated with this output destination are complete, and the last log file is closed (see Section 3.1.6.3), the Log Service changes the configuration file name to:

<filename>_<closetime>.cfg

so that a log reader can know that this configuration file is no longer active, and that the configuration specified is associated with one or more log files with the same <filename> prefix and qualifying <closetime> suffix.

3.1.6.3 Log File Naming Rules

The content of a log file (or files) conforms to the configuration expressed in the <filename>.cfg file.

There are two notable moments in the life of a log file (or files), which correspond to a name change of the log file.

1. When a log file is created or is in use, the log file has the file name:

   <filename>_<createtime>.log

2. When a log file is closed, the log file has the file name:

   <filename>_<createtime>_<closetime>.log

A log file can close for one of these reasons:

- An application log stream is closed by its last user, or
- The last application log stream user dies (which is an implicit log stream close).
- A log file has reached maximum capacity and a log file full action is undertaken; specifically halt or rotate.

A closed log file does not imply a closed log stream. First, the constant log streams (notification, alarm, and system) are always available. Second, in future versions of this specification there may be several independent output destinations associated with the same log stream as suggested by the log stream handler concept (see Section 2.3).

The log file naming rules for the various log full actions is now considered.

If an application log stream is closed, or when the *LogFullAction* is either *ROTATE* or *HALT* and the log file has reached the configure *MAX_FILE_SIZE*, the file is given its file closed name.

In the case of *ROTATE*, a new file is created with a <createtime> that is the same as the <closetime> of the just-finished log file. This makes the ordered creation of these files simple to identify.

In the case of *WRAP*, there is only ever a single file that is never finished and so its name is never augmented with a <closetime>. The exception is when this file is associated with an application log stream and the log stream is closed.

the format of <createtime> and <closetime> is:

```
yyyymmdd__hhmmss
```

This order allows for easy lexicographical sorting by date and time of any group of files.

So a completed *ROTATE* log file might read:

```
myLogFile_20050712_102316__20050713_030854.log
```

3.1.6.4 Log File Behaviors

Log records must be readable immediately after they are written to a log file. A file reader cannot be blocked from accessing a file that is currently being written to.

Log records are written to a file in the order in which they arrive at the output destination (as opposed to the order of its time-stamp). Third party reader tools can use the time-stamp value of each log record if the temporal sequence is desired.

All log records are given an ascending 32 bit record-id value per distinct output destination (in this case a log file) that is assigned in the order in which the log record arrived at the particular output destination.

It is left as an implementation matter as to if, how or when log files can be deleted, moved, compacted, archived or otherwise modified in a running system while the log stream is active and how these activities are coordinated with the Log Service. Log Service operations to cover such cases may be introduced in future revisions of this document.

3.1.7 Internationalization

Internationalization refers to a means by which the text associated with a log record is formatted and presented in the preferred language of choice to a human reader. The

SA Forum Notification (NTF) Service[2] data type *saNtfClassIdT* provides the principle data points that allow for a catalog lookup of the substitute values necessary to achieve a specific language presentation.

Though the Log service provides the data points to support Internationalization, the actual method for achieving it is postponed to some future Log Service release.

3.2 Include File and Library Names

The following statement containing declarations of data types and function prototypes must be included in the source of an application using the Log Service API:

#include <saLog.h>

To use the Log Service API, an application must be bound with the following library:

libSaLog.so

3.3 Type Definitions

The Log Service uses the types described in the following sections.

3.3.1 Handles

3.3.1.1 SaLogHandleT

typedef SaUint64T SaLogHandleT;

The type of the handle supplied by the Log Service to a process during initialization of the Log Service and used by a process when it invokes functions of the Log Service API so that the Log Service can recognize the process.

3.3.1.2 SaLogStreamHandleT

typedef SaUint64T SaLogStreamHandleT;

The type of the handle associated with a particular log stream.

3.3.2 Log Types

3.3.2.1 Log Stream Names

The following log stream name constants map to the three well-known log streams.

```
#define SA_LOG_STREAM_SYSTEM                "safLgStr=saLogSystem"
#define SA_LOG_STREAM_NOTIFICATION          "safLgStr=saLogNotification"
#define SA_LOG_STREAM_ALARM                 "safLgStr=saLogAlarm"
```

These log stream name constant values have the following interpretation:

- *SA_LOG_STREAM_ALARM* - this log stream name is used by the SA Forum Notification (NTF) Service [2] to open the alarm log stream, which tracks to the ITU specifications alarm reporting (X.733) and security alarm reporting (X.736). There is one alarm log stream in a cluster.
- *SA_LOG_STREAM_NOTIFICATION* - this log stream name is used by the SA Forum Notification (NTF) Service [2] to open the notification log stream, which tracks to the ITU specifications object management (X.730) and state management (X.731). There is one notification log stream in a cluster.
- *SA_LOG_STREAM_SYSTEM* - this log stream name is used by applications to open the system log stream in order to log circumstances that are system relevant, but less formal than alarm or notification logging. These log records are noteworthy or supplementary to a reasonable view of a cluster's (historic) circumstances. There is one system log stream in a cluster.

Application log stream names are user defined and must be cluster wide unique. As such, no application log stream constant names are identified in this specification. There can be any number of application log streams in a cluster.

3.3.2.2 SaLogSeverityT and SaLogSeverityFlagsT

The *SaLogSeverityT* and *SaLogSeverityFlagsT* types are used to express severity in the context of applications and system log records and log streams.

```
#define SA_LOG_SEV_EMERGENCY   0
#define SA_LOG_SEV_ALERT       1
#define SA_LOG_SEV_CRITICAL    2
#define SA_LOG_SEV_ERROR       3
#define SA_LOG_SEV_WARNING     4
#define SA_LOG_SEV_NOTICE      5
#define SA_LOG_SEV_INFO        6

typedef SaUint16T SaLogSeverityT;

typedef SaUint16T SaLogSeverityFlagsT;
```

SaLogSeverityT is a used to specify the severity level of a particular system or application log record (see Section 3.3.5.3) when the *saLogWriteLog()* is invoked (see Section 3.5.3).

SaLogSeverityFlagsT is a bitmap used in the *SaLogFilterSetCallbackT()* callback (see Section 3.5.5). In this case, each *SA_LOG_SEV_* value identifies a bit position in the *SaLogSeverityFlagsT* bitmap to allow (bit is 1) or disallow (bit is 0) log records of a particular severity on to the associated system or application log stream.

These severity levels have the following interpretation [7]:

- *SA_LOG_SEV_EMERGENCY* - the system is unusable
- *SA_LOG_SEV_ALERT* - action must be taken immediately
- *SA_LOG_SEV_CRITICAL* - critical conditions
- *SA_LOG_SEV_ERROR* - error conditions
- *SA_LOG_SEV_WARNING* - warning conditions
- *SA_LOG_SEV_NOTICE* - normal but significant condition
- *SA_LOG_SEV_INFO* - informational messages

3.3.2.3 SaLogBufferT

```
typedef struct {
        SaSizeT     logBufSize;
        SaUint8T    *logBuf;
} SaLogBufferT;
```

This data structure contains the body of the log record and is provided while invoking the *saLogWriteLog()* function. The Log Service does not interpret or parse the interior of a *SaLogBufferT*, except as implied by either the @Cb or @Ci format tokens (see Section 3.1.5.1) when used in a format expression (see Section 3.1.5.2).

3.3.2.4 SaLogAckFlagsT

The *SaLogAckFlagsT* type is used in the *saLogWriteLogAsync()* calls. A parameter of the type *SaLogAckFlagsT* indicates the kind of the required acknowledgment:

```
#define SA_LOG_RECORD_WRITE_ACK 0x1
typedef SaUint32T SaLogAckFlagsT;
```

SA_LOG_RECORD_WRITE_ACK - indicates that the calling logger requires an acknowledgment to confirm whether the log record could be written to the destination output log file associated with the log stream. If *SA_LOG_RECORD_WRITE_ACK* is not set, the calling logger does not require an acknowledgment.

3.3.2.5 SaLogStreamOpenFlagsT

The following values specify the open attributes used in the *saLogStreamOpen()* while opening an application log stream.

```
#define SA_LOG_STREAM_CREATE     0x1
typedef SaUint8T SaLogStreamOpenFlagsT;
```

A value or parameter of the type *SaLogStreamOpenFlagsT* is zero or the bitwise OR of the values in the following list:

- *SA_LOG_STREAM_CREATE* - This flag requests the creation of an application log stream if the identified log stream does not already exist.

3.3.3 Log Service API and Notification Types

The Log Service API interface uses SA Forum Notification (NTF) Service [2] data types *SaLogNtfLogHeaderT* (see Section 3.3.5.2) and *SaLogGenericLogHeaderT* (see Section 3.3.5.3) as part of their definition. In order to resolve these data types, the *saLog.h* file simply includes the SA Forum Notification (NTF) Service [2] header file, as follows:

#include <saNtf.h>

3.3.4 Log Service as Notification Producer

The Log Service is also a producer of Notifications (see Section 4). The values placed in certain fields within notifications are assigned by the Log Service.

3.3.4.1 SaLogNtfIdentifiersT

The Log Service defines a set of Notification identifiers, which are scoped to the Log Service only.

```
typedef enum {
        SA_LOG_NTF_LOGFILE_PERCENT_FULL= 1 /* used in capacity alarm */
} SaLogNtfIdentifiersT;
```

3.3.4.2 SaLogNtfAttributesT

The object change notifications allow a list of attributes to be delivered. The Log Service notifications that have such a list are:

- log stream create
- log stream destroy

```
typedef enum {
        SA_LOG_NTF_ATTR_LOG_STREAM_NAME = 1,
        SA_LOG_NTF_ATTR_LOGFILE_NAME,
        SA_LOG_NTF_ATTR_LOGFILE_PATH_NAME
} SaLogNtfAttributesT;
```

3.3.5 Log Record Types

3.3.5.1 SaLogHeaderTypeT

```
typedef enum {
        SA_LOG_NTF_HEADER          = 1,
        SA_LOG_GENERIC_HEADER      = 2
} SaLogHeaderTypeT;
```

The values of the *SaLogHeaderTypeT* have the following interpretations:

- *SA_LOG_NTF_HEADER* - The log record header structure used for an *saLogWriteLog()* is *SaLogNtfLogHeaderT*, which is suitable for the alarm or notification log streams.
- *SA_LOG_GENERIC_HEADER* - The log record header structure used for a *saLogWriteLog()* is *SaLogGenericLogHeaderT*, which is suitable for the system or any application log stream.

3.3.5.2 SaLogNtfLogHeaderT

```
typedef struct {
        SaNtfIdentifierT      notificationId;
        SaNtfEventTypeT       eventType;
        SaNameT               *notificationObject;
        SaNameT               *notifyingObject;
        SaNtfClassIdT         *notificationClassId;
        SaTimeT               eventTime;
} SaLogNtfLogHeaderT;
```

This structure contains the fields specific to a notification or alarm log record header. It must be populated by the logger when *saLogWriteLog()* is invoked. The fields have the following interpretation:

- *notificationId* - (defined in saNtf.h [2]). This is a cluster-wide unique identifier value provided to the Log Service by a Notification service client. This field may be set to *SA_NTF_IDENTIFIER_UNUSED* [2] if no identifier is provided. The Log Service does not police this value for uniqueness.
- *eventType* - (defined in saNtf.h [2]) This field must be set. It reflects the event type of the notification. This value is achieved by OR-ing together an enum value of type *SaLogNtfEventTypeT* with an enum value of type

SaLogNtfNotificationTypeT. This produces a two-part value that expresses the macro type of the event like alarm, security alarm, object, state or attribute change as well as the exact event subtype such as an alarm that is QOS or environment related.

- *notificationObject* - A non-NULL pointer to the name of the logical entity about which the notification is generated, identified by its full LDAP name.
- *notifyingObject* - This field must be set. A non-NULL pointer to the name of the logical entity that is sending the notification, identified by its full LDAP name.
- *notificationClassId* - This field is optional (defined in saNtf.h [2]). It uniquely identifies the kind of situation that caused the notification. This identifier alone is sufficient to unequivocally identify the kind of situation, no other information from the notification is necessary.
- *eventTime* - This field must be set. This field contains the time at which an event is detected. This may not be the same time at which the event was reported or the notification was logged.

3.3.5.3 SaLogGenericLogHeaderT

```
typedef struct {
        SaNtfClassIdT       *notificationClassId;
        const SaNameT       *logSvcUsrName;
        SaLogSeverityT      logSeverity;
} SaLogGenericLogHeaderT;
```

This structure contains the fields that go into a log record header and whose destination is either the system or an application specific log stream. The fields have the following interpretation:

- *notificationClassId* - (defined in saNtf.h) This field is used for internationalization. This is an optional field that may be set to NULL. The Log Service itself just passes this value through to the output destination. Future versions of this specification will address internationalization issues (see Section 3.1.7).
- *logSvcUsrName* - The LDAP name used by the logger to identify itself. This will typically be a component or service unit provided the user is a component under the control of the Availability Management Framework. This argument only needs to be specified on a per log record basis in the *saLogWriteLog()* or *saLogWriteLogAsync()* API when the logger wants to override the default user name maintained by the Log Service on behalf of a logger. The default user name is fetched by the Log Service library from the *SA_AMF_COMPONENT_NAME* environment variable by using a POSIX *getenv()* subroutine. This mechanism avoids cross-library dependencies. If this argument is not specified at *saLogWriteLog()* time and the environment variable is not set, it is an error.
- *logSeverity* - This field must be set to a single severity level value for this log record. The various severity levels supported by the Log Service are defined in Section 3.3.2.2.

3.3.5.4 SaLogHeaderT

```
typedef union {
        SaLogNtfLogHeaderT          ntfHdr;
        SaLogGenericLogHeaderT      genericHdr;
} SaLogHeaderT;
```

The *SaLogHeaderT* type contains log record header information that is specific to the log stream for which the log record is destined. If the log record is destined for either the notification or alarm log streams then the *ntfHdr* structure must be properly populated (refer to Section 3.3.5.2). If the log record is destined for either the system or an application log stream then *genericHdr* must be properly populated (refer to Section 3.3.5.3).

3.3.5.5 SaLogRecordT

The following data structure describes the contents of a log record. This data-structure wraps data structures that have been described earlier.

```
typedef struct {
        SaTimeT                 logTimeStamp;
        SaLogHeaderTypeT        logHdrType;
        SaLogHeaderT            logHeader;
        SaLogBufferT            *logBuffer;
} SaLogRecordT;
```

The fields in this data structure have the following interpretation:

- *logTimeStamp* - This field contains the time at which the log is produced. If the time-stamp can not be provided by the user then the constant *SA_TIME_UNKNOWN* shall be specified instead, which means the Log Service needs to supply the time-stamp.
- *logHdrType* - This field must be set. It indicates the log record header type that is populated in the union *SaLogHeaderT* (see Section 3.3.5.4) of the next parameter, *logHeader*.
- *logHeader* - Refer to Section 3.3.5.4 for details on how to populate this field based on the *logHdrType* field.
- *logBuffer* - Contains the body of the log record, which the Log service treats as a single opaque data unit. It may be NULL indicating that there is no body. The Log Service transfers the log body as a part of the log record reliably through the log stream to its final output destination where this data unit is subject to either the @Cb or @Ci format tokens (see Section 3.1.5.1) both of which result in only printable character output.

3.3.6 Application Log Types

This section describes additional data-structures used by application loggers only.

3.3.6.1 SaLogFileFullActionT

```
typedef enum {
        SA_LOG_FILE_FULL_ACTION_WRAP        = 1,
        SA_LOG_FILE_FULL_ACTION_HALT        = 2,
        SA_LOG_FILE_FULL_ACTION_ROTATE      = 3
} SaLogFileFullActionT;
```

This type explains Log Service behavior when a file's maximum log size is reached. This policy is specified while opening a new application log stream. These policies are as follows:

SA_LOG_FILE_FULL_ACTION_WRAP - Once the maximum log file size has been reached, the oldest log records are deleted as needed to allow for new log records.

SA_LOG_FILE_FULL_ACTION_HALT - The log file is full. No more log records are allowed in this log file.

SA_LOG_FILE_FULL_ACTION_ROTATE - When the current log file is full, a new log file is created (with <createtime>) to which future log records are now written.

3.3.6.2 SaLogFileCreateAttributesT

```
typedef struct {
        SaStringT               *logFileName;
        SaStringT               *logFilePathName;
        SaUint64T               maxLogFileSize;
        SaUint32T               maxLogRecordSize;
        SaBoolT                 haProperty;
        SaLogFileFullActionT    logFileFullAction;
        SaUint16T               maxFilesRotated;
        SaStringT               *logFileFmt;
} SaLogFileCreateAttributesT;
```

This type contains the log file creation information that needs to be supplied while creating a new application log stream. The fields are interpreted as follows:

- *logFileName* - The POSIX log file name to be associated with an application specific log stream. A value must be set.
- *logFilePathName* - The POSIX path, which qualifies where the log file resides in the cluster. Details regarding where log files can live and how location within a cluster is actually specified is implementation-specific.
- *maxLogFileSize* - The maximum size a log file may grow to, in bytes. A value of zero indicates no predefined limit. If the specified limit is exceeded the *logFileFullAction* action as described in Section 3.3.6.1 is invoked. A value must be set.
- *maxLogRecordSize* - This is required. Max log record size that can be written to this file. Log records larger than this size shall be truncated. A value must be set.
- *haProperty* - Indicates if the log file must always be available and implies file replication and persistency (see Section 3.1.6.1). A value must be set.
- *logFileFullAction* - explains the Log Service behavior when a file's maximum log size in bytes is reached. Refer to Section 3.3.6.1 for details. A value must be set.
- *maxFilesRotated* - Indicates the number of files maintained at a time if the *logFileFullAction* policy is chosen as *SA_LOG_FILE_FULL_ACTION_ROTATE.* If the *logFileFullAction* policy is not *SA_LOG_FILE_FULL_ACTION_ROTATE,* this field is ignored by the Log Service.
- *logFileFmt* - contains a log record format expression specified by the logger. If this value is NULL, then the Log Service uses the default format expression for the target log stream type (see Section 3.1.5.3).

3.3.7 SaLogCallbacksT

The *SaLogCallbacksT* structure is defined as follows:

```
typedef struct {
        SaLogFilterSetCallbackT      saLogFilterSetCallback;
        SaLogStreamOpenCallbackT     saLogStreamOpenCallback;
        SaLogWriteLogCallbackT       saLogWriteLogCallback;
} SaLogCallbacksT;
```

This structure contains the callback function pointers supplied by the logger to the Log Service. The Log Service will invoke these callbacks at well defined moments.

3.4 Library Life Cycle

3.4.1 saLogInitialize()

Prototype

```
SaAisErrorT saLogInitialize(
        SaLogHandleT             *logHandle,
        const SaLogCallbacksT    *logCallbacks,
        SaVersionT               *version
);
```

Parameters

logHandle - [out] A pointer to the handle designating this particular initialization of the Log Service that is to be returned by the Log Service.

*logCallback*s - [in] If *logCallbacks* is set to NULL, no callback is registered; otherwise, it is a pointer to a *SaLogCallbacksT* structure, containing the callback functions of the process that the Log Service may invoke. Only non-NULL callback functions in this structure will be registered.

version - [in/out] As an input parameter, *version* is a pointer to the required Log Service version. In this case, *minorVersion* is ignored and should be set to 0x00.
As an output parameter, the version actually supported by the Log Service is delivered.

Description

This function initializes the Log Service for the invoking process and registers the various callback functions. This function must be invoked prior to the invocation of any other Log Service functionality. The handle *logHandle* is returned as the reference to this association between the process and the Log Service. The process uses this handle in subsequent communication with the Log Service.

If the implementation supports the required *releaseCode,* and a major version >= the required *majorVersion*, SA_AIS_OK is returned. In this case, the *version* parameter is set by this function to:

- *releaseCode* = required release code
- *majorVersion* = highest value of the major version that this implementation can support for the required *releaseCode*

- *minorVersion* = highest value of the minor version that this implementation can support for the required value of *releaseCode* and the returned value of *majorVersion*

If the above mentioned condition cannot be met, SA_AIS_ERR_VERSION is returned, and the *version* parameter is set to:

```
if (implementation supports the required releaseCode)
    releaseCode = required releaseCode
else {
    if (implementation supports releaseCode higher than the required
    releaseCode)
        releaseCode = the least value of the supported release codes that is
        higher than the required releaseCode
    else
        releaseCode = the highest value of the supported release codes that is
        less than the required releaseCode
}
```

majorVersion = highest value of the major versions that this implementation can support for the returned *releaseCode*

minorVersion = highest value of the minor versions that this implementation can support for the returned values of *releaseCode* and *majorVersion*

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the Log Service provider is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).

SA_AIS_ERR_VERSION - The *version* parameter is not compatible with the version of the Log Service implementation.

See Also

saLogSelectionObjectGet(), saLogDispatch(), saLogFinalize()

3.4.2 saLogSelectionObjectGet()

Prototype

```
SaAisErrorT saLogSelectionObjectGet(
        SaLogHandleT              logHandle,
        SaSelectionObjectT        *selectionObject
);
```

Parameters

logHandle - [in] The handle, obtained through the *saLogInitialize()* function, designating this particular initialization of the Log Service.

selectionObject - [out] A pointer to the operating system handle that the process can use to detect pending callbacks.

Description

This function returns the operating system handle, *selectionObject,* associated with the handle *logHandle*. The invoking process can use this handle to detect pending callbacks, instead of repeatedly invoking *saLogDispatch()* for this purpose.

In a POSIX environment, the operating system handle is a file descriptor that is used with the *poll()* or *select()* system calls to detect incoming callbacks.

The *selectionObject* returned by *saLogSelectionObjectGet()* is valid until *saLogFinalize()* is invoked on the same handle *logHandle*.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the Log Service provider is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).

See Also

saLogInitialize(), saLogDispatch(), saLogFinalize()

3.4.3 saLogDispatch()

Prototype

```
SaAisErrorT saLogDispatch(
        SaLogHandleT            logHandle,
        SaDispatchFlagsT        dispatchFlags
);
```

Parameters

logHandle - [in] The handle, obtained through the *saLogInitialize()* function, designating this particular initialization of the Log Service.

dispatchFlags - [in] Flags that specify the callback execution behavior of the *saLogDispatch()* function, which have the values *SA_DISPATCH_ONE*, *SA_DISPATCH_ALL*, or *SA_DISPATCH_BLOCKING*, as defined in the SA Forum Overview document.

Description

This function invokes, in the context of the calling thread, pending callbacks for the handle *logHandle* in a way that is specified by the *dispatchFlags* parameter.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INVALID_PARAM - The *dispatchFlags* parameter is invalid.

See Also

saLogInitialize(), saLogSelectionObjectGet()

3.4.4 saLogFinalize()

Prototype

```
SaAisErrorT saLogFinalize(
        SaLogHandleT        logHandle
);
```

Parameters

logHandle - [in] The handle, obtained through the *saLogInitialize()* function, designating this particular initialization of the Log Service.

Description

The *saLogFinalize()* function closes the association, represented by the *logHandle* parameter, between the invoking process and the Log Service. The process must have invoked *saLogInitialize()* before it invokes this function. A process must invoke this function once for each handle acquired by invoking *saLogInitialize()*.

If the *saLogFinalize()* function returns successfully, the *saLogFinalize()* function releases all resources acquired when *saLogInitialize()* was called. Moreover, it closes all log streams that are still open for the particular handle. Furthermore, it cancels all pending callbacks related to the particular handle.

After *saLogFinalize()* is called, the selection object is no longer valid. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

See Also

saLogInitialize()

3.5 Log Service Operations

3.5.1 saLogStreamOpen() and saLogStreamOpenAsync()

Prototype

```
SaAisErrorT saLogStreamOpen(
        SaLogHandleT                    logHandle,
        SaNameT                         logStreamName,
        SaLogFileCreateAttributesT      *logFileCreateAttributes,
        SaLogStreamOpenFlags            logStreamOpenFlags,
        SaTimeT                         timeout,
        SaLogStreamHandleT              *logStreamHandle
);

SaAisErrorT saLogStreamOpenAsync(
        SaLogHandleT                    logHandle,
        SaNameT                         logStreamName,
        SaLogFileCreateAttributesT      *logFileCreateAttributes,
        SaLogStreamOpenFlags            logStreamOpenFlags,
        SaInvocationT                   invocation
);
```

Parameters

logHandle - [in] The handle, obtained through the *saLogInitialize()* function, designating this particular initialization of the Log Service.

logStreamName - [in] This parameter designates the DN name of the log stream to open. This may be one of the well-known log stream names (see Section 3.3.2.1) or it may be a user defined cluster wide unique application log stream name.

logFileCreateAttributes - [in] A pointer to the *SaLogFileCreateAttributesT* (see Section 3.3.6.2) that describes the attributes associated with an application log stream only. If one of the well-known log streams is being opened this should be NULL. Other considerations are as follows:

- If the intent is only to open an existing application log stream by supplying the same *logStreamName*, then this value should be NULL.
- If the intent is to open and create an application log stream that does not yet exist, then *logFileCreateAttributes* must be populated and its pointer passed.
- If the intent is to open a (possibly) existing application log stream, but still specify creation attribute values, then the provided values must be identical to those values provided by the initial logger who successfully created the application log stream.

logStreamOpenFlags - [in] The value of the parameters is constructed by a bit OR of the flags defined by the (see Section 3.3.2.5). This value is only set when opening an application log stream. If one of the well-known log streams is being opened this must not be set. Other considerations are as follows:

- If the intent is only to open an existing application log stream by supplying the same *logStreamName*, then this value may not be set.
- If the intent is to open and create an application log stream that does not yet exist, then the *SA_LOG_STREAM_CREATE* flag must be set.
- If the intent is to open a (possibly) existing application log stream by providing an identical set of values in the parameter *logFileCreateAttributes*, then the *SA_LOG_STREAM_CREATE* flag must also be set.

timeout - [in] The *saLogStreamOpen()* invocation is considered to have failed if it does not complete by the time specified. A log stream may still be created in such a case, as the outcome is non-deterministic.

invocation - [in] This parameter allows the invoking logger to match this invocation of *saLogStreamOpenAsync()* with the corresponding *(*SaLogStreamOpenCallbackT)()* callback call.

*logStreamHandl*e- [out] A pointer to the log stream handle, allocated in the address space of the invoking process. If the log stream is opened successfully, the Log Service stores in *logStreamHandle* the handle that the logger uses to access the correct log stream in subsequent invocations of the functions of the Log Service Operations APIs.

Description

The *saLogStreamOpen()* opens a log stream. If the log stream is an application log stream and the named application log stream does not exist, then the *logFileCreateAttributes* must be populated and passed and the *SA_LOG_STREAM_CREATE* flag is set in the *logStreamOpenFlags* parameter.

For the three well-known log streams, the *logStreamHandle* references the existing alarm, notification, or system log streams, which are created when the Log Service is initialized in the cluster. These log streams persist over the life time of the Log Service in the cluster.

An invocation of *saLogStreamOpen()* is blocking. If the log stream is successfully opened, a new log stream handle is returned upon completion. A log stream can be opened multiple times from within the same process or by different processes.

Completion of the *saLogStreamOpenAsync()* function is signaled by an invocation of the associated *SaLogStreamOpenCallbackT()* callback function, which must have been supplied when the process invoked the *saLogInitialize()* call. The process supplies the value of *invocation* when it invokes the *saLogStreamOpenAsync()* function and the Log Service gives that value of *invocation* back to the application when it invokes the corresponding *SaLogStreamOpenCallbackT()* function. The *invocation* parameter is a mechanism that enables the process to determine which call triggered which callback.

Application log streams have a default log record format expression associated with them as described in Section 3.1.5.3. If this format expression is not desired a different format may be specified while creating the log stream using the syntax described in Section 3.1.5. Once a format expression is associated with a log stream it can not be changed over the life of the log stream.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred, or the timeout, specified by the *timeout* parameter, occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with *saLogInitialize()* was incomplete, since the *saLogStreamOpenCallbackT()* callback function is missing. This applies only to the *saLogStreamOpenAsync()* function.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this error is returned for each of the following cases:

- An application log stream is identified, and the *SA_LOG_STREAM_CREATE* flag is set in *logStreamOpenFlags* but the *logFileCreateAttributes* parameter is NULL.
- An application log stream is identified, and the *SA_LOG_STREAM_CREATE* flag is not set in *logStreamOpenFlags* but the *logFileCreateAttributes* parameter is not NULL.
- The *LogStreamName* is not a DN, or the type of its first RND is not *safLgStr*.

SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the Log Service provider is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).

SA_AIS_ERR_NOT_EXIST - The *SA_LOG_STREAM_CREATE* flag is not set and the *logFileCreateAttributes is NULL* and the application log stream designated by *logStreamName* does not exist.

SA_AIS_ERR_EXIST - The application log stream designated by *logStreamName* already exists and the *logFileCreateAttributes* is either non-NULL, or the values provided do not match the values used to originally open this application log stream.

SA_AIS_ERR_BAD_FLAGS - The *logStreamOpenFlags* parameter is invalid.

See Also

saLogStreamClose()

3.5.2 SaLogStreamOpenCallbackT

Prototype

```
typedef void (*SaLogStreamOpenCallbackT)(
        SaInvocationT           invocation,
        SaLogStreamHandleT      logStreamHandle,
        SaAisErrorT             error
);
```

Parameters

invocation - [in] This parameter was supplied by a process in the corresponding invocation of the *saLogStreamOpenAsync()* function and is used by the Log Service in this callback. This invocation parameter allows the process to match the invocation of that function with this callback.

logStreamHandle - [in] The handle that designates the log stream if *error* is SA_AIS_OK.

error - [in] This parameter indicates whether the *saLogStreamOpenAsync()* function was successful. The values that can be returned are:

- SA_AIS_OK - The function completed successfully.
- SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.
- SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.
- SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may try again.
- SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the provider of the service is out of memory and cannot provide the service.
- SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).
- SA_AIS_ERR_NOT_EXIST - The log stream, identified by *logStreamName*, does not exist, and the value of the SA_LOG_STREAM_CREATE flag is not set.
- SA_AIS_ERR_EXIST - The log stream already exists and the *logFileCreateAttribs* creation attributes are different from the ones used at creation time.
- SA_AIS_ERR_BAD_FLAGS - The *logStreamOpenFlags* parameter is invalid.

Description

The Log Service calls this callback function when the operation requested by the invocation of *saLogStreamOpenAsync*() completes. This callback is invoked in the context of a thread issuing an *saLogDispatch()* call on the handle *logHandle*, which was specified in the *saLogStreamOpenAsync*() call. If successful, the reference to the opened/created stream is returned in *logStreamHandle*; otherwise, an error is returned in the error parameter.

Return Values

None

See Also

saLogStreamOpenAsync(), saLogDispatch(), saLogInitialize()

3.5.3 saLogWriteLog() and saLogWriteLogAsync()

Prototype

```
SaAisErrorT saLogWriteLog(
        SaLogStreamHandleT      logStreamHandle,
        SaTimeT                 timeOut,
        SaLogRecordT            *logRecord
);

SaAisErrorT saLogWriteLogAsync(
        SaLogStreamHandleT      logStreamHandle,
        SaInvocationT           invocation,
        SaLogAckFlagsT          ackFlags,
        SaLogRecordT            *logRecord
);
```

Parameters

logStreamHandle - [in] The handle that designates the destination log stream for this log record. The handle *logStreamHandle* must have been obtained previously by the invocation of the s*aLogStreamOpen()* or s*aLogStreamOpenAsync()* function.

timeOut - [in] The *saLogWriteLog()* invocation is considered to have failed if it does not complete by the time specified. A log record may be still written to the log stream.

ackFlags - [in] The kind of the required acknowledgment. This field must be set to zero or to *SA_LOG_RECORD_WRITE_ACK*. In the latter case, the caller requires to be acknowledged whether the log record can be logged. If set to 0 no such acknowledgement is desired.

logRecord - [in] A non-NULL pointer to the contents of the log record. The various fields of this parameter are described in details in Section *3.3.5.5*. Refer to that section for a detailed overview of how the log record needs to be populated, including which fields are required and which are optional.

invocation - [in] This parameter associates this invocation of *saLogWriteLogAsync()* with a corresponding invocation of the *SaLogWriteLogCallbackT()* function. This parameter is ignored if *ackFlags* is set to zero, meaning that the *SaLogWriteLogCallbackT()* function is not called, and the caller is not informed whether an error occurred.

Description

This API is used to log a record designated by *logRecord* to a stream specified by the *logStreamHandle*.

An invocation of *saLogWriteLog()* is blocking. The log record is written to the log file associated with the stream designated by *logStreamHandle* upon successful completion.

An invocation of *saLogWriteLogAsync()* is non-blocking. Completion of the *saLogWriteLogAsync()* signifying that a log record has been written to the log file associated with the stream designated by *logStreamHandle* is optionally signaled by an invocation of the *SaLogWriteLogCallbackT()* callback function if the flag *SA_LOG_RECORD_WRITE_ACK* is set in the *ackFlags.*

Each log record written to a log file is an atomic operation so that concurrent writes must be properly handled.

If the destination log file has reached maximum capacity and the *logFileFullAction* policy is *SA_LOG_FILE_FULL_ACTION_HALT* then a SA_AIS_ERR_NO_RESOURCES error code is returned.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred, or the timeout, specified by the *timeOut* parameter, occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logStreamHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saLogStreamOpen()* or *saLogStreamOpenCallback()* functions, or the corresponding log stream has already been closed.
- The handle *logHandle* that was passed to the *saLogStreamOpen()* or *saLogStreamOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with *saLogInitialize()* was incomplete, since the *SaLogWriteLogCallbackT()* callback function is missing. This applies only to the *saLogWriteLogAsync()* function if *SA_LOG_RECORD_WRITE_ACK* flag is set in the *ackFlags*.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this error is returned for each of the following cases:

- The log record type designated by *logHdrType* in *SaLogRecordT* does not correspond to the type of log stream implied by *logStreamHandle*.
- The *logSvcUsrName* (see Section 3.3.5.3) is not provided and the *SA_AMF_COMPONENT_NAME* environment variable is not properly set.

SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the Log Service provider is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory), including the case that the destination log file associated with the stream designated by *logStreamHandle* has reached maximum capacity and the *logFileFullAction* policy is *SA_LOG_FILE_FULL_ACTION_HALT.*

SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

See Also

saLogStreamOpen(), saLogStreamOpenAsync(), SaLogWriteLogCallbackT

3.5.4 SaLogWriteLogCallbackT

Prototype

```
typedef void (*SaLogWriteLogCallbackT)(
        SaInvocationT      invocation,
        SaAisErrorT        error
);
```

Parameters

invocation - [in] This parameter associates an invocation of *saLogWriteLogAsync()* with a corresponding invocation of the *SaLogWriteLogCallbackT()* function.

error - [in] This parameter indicates whether the *saLogWriteLogAsync()* function was successful. The values that can be returned are:

- SA_AIS_OK - The function completed successfully.
- SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

- SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.
- SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.
- SA_AIS_ERR_NO_MEMORY - Either the Log Service library or the Log Service provider is out of memory and cannot provide the service.
- SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory), including the case that the destination log file associated with the stream designated by *logStreamHandle* in the corresponding invocation of the *saLogWriteLogAsync()* function has reached maximum capacity and the *logFileFullAction* policy is *SA_LOG_FILE_FULL_ACTION_HALT.*
- SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

Description

The Log Service calls this callback function when the operation requested by the invocation of *saLogWriteLogAsync*() completes or fails, provided a desire for receiving such an acknowledgement was indicated by setting the *SA_LOG_RECORD_WRITE_ACK* flag in the *ackFlags* field during the *saLogWriteLogAsync()* function invocation.

This callback is invoked in the context of a thread issuing an *saLogDispatch()* call on the handle *logHandle*, which was obtained by the invocation of *saLogInitialize()* function. If successful, the log record is written to the destination log file associated with the log stream designated by *logStreamHandle* in *saLogWriteLogAsync()* function.

Return Values

None

See Also

saLogWriteLogAsync(), saLogDispatch(), saLogInitialize()

3.5.5 SaLogFilterSetCallbackT

Prototype

```
typedef void (*SaLogFilterSetCallbackT)(
        SaLogStreamHandleT      logStreamHandle,
        SaLogSeverityFlagsT     logSeverity
);
```

Parameters

logStreamHandle - [in] The handle that designates either the well-known system log stream or one of the application log streams. This handle was obtained previously by the invocation of the s*aLogStreamOpen()* or s*aLogStreamOpenAsync()* function.

logSeverity - [in] explains which log records are allowed to be forwarded from a logger source. This is a bitmap that describes the severity levels at which logging is enabled, i.e., only log records with severity levels enabled in the *logSeverity* will be forwarded to the Log Service.

Description

The Log Service invokes this callback to request the process to log at only the levels indicated in the bitmap designated by *logSeverity* for the log stream associated with the logStreamHandle. Only the system and application log streams use *logSeverity*. By default, log records with all severity levels are allowed and the Log Service does not filter any log records based on the severity level.

Once the *logSeverity* bitmap arrives, loggers should not produce log records with severities that are disabled. However, if a logger does produce such log records or this logger did not provide this callback function, the Log Service always monitors the severity levels of the log records introduced by way of *saLogWriteLog()* and will not ignore log records that are not allowed on the log stream.

This callback may be invoked as a consequence of an administrative operation to set a particular log steam at desired severity levels or as a matter of initial configuration, which causes a pre-configured *logSeverity* to be pushed to the affected processes that are link with the Log Service library.This callback can happen any time after a successful completion of *saLogStreamOpen()* or the *(*SaLogStreamOpenCallbackT)()* callback.

The most recent *logSeverity* is the one that is honored, i.e., the *logSeverity* delivered by the last invocation of this callback displaces the *logSeverity* delivered in the previous callback.

Return Values

None

See Also

saLogInitialize()

3.5.6 saLogStreamClose()

Prototype

SaAisErrorT saLogStreamClose(

SaLogStreamHandleT logStreamHandle

);

Parameters

logStreamHandle - [in] The handle that designates the log stream that needs to be closed. The handle *logStreamHandle* must have been obtained previously by the invocation of the s*aLogStreamOpen()* or s*aLogStreamOpenAsync()* function.

Description

The invocation of this API closes the log stream designated by *logStreamHandle,* which was opened by an earlier invocation of the *saLogStreamOpen()* or *saLogStreamOpenAsync()* function*.*

After this invocation, the handle *logStreamHandle* is no longer valid.

When the invocation of the *saLogStreamClose()* function completes successfully, and if it is an application log stream, and no other process has that application log stream open, then the log file associated with that application log stream is closed and renamed with a <closetime> that indicates when the last user of the log stream designated by *logStreamHandle* closed the stream (see 3.1.6.4).

Closing a log stream frees all resources allocated by the Log Service for this process.

If a process terminates, the Log Service implicitly closes all log streams that are open for this process.

This call cancels all pending callbacks that refer directly or indirectly to the handle *logStreamHandle*. Note that as the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.

SA_AIS_ERR_BAD_HANDLE - The handle *logStreamHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saLogStreamOpen()* or *saLogStreamOpenAsync()* functions, or the corresponding log stream has already been closed.
- The handle *logHandle* that was passed to the *saLogStreamOpen()* or *saLogStreamOpenAsync()* functions has already been finalized.

See Also

saLogStreamOpen(), saLogStreamAsync()

4 Administrative API

4.1 Log Service Administration API Model

4.1.1 Log Service Administration API Basics

This section describes the various administrative API functions that the IMM Service exposes on behalf of the Log Service to a system administrator. These API functions are described using a ‘C’ API syntax. The main clients of this administrative API are system management applications, SNMP agents and CIM providers that typically convert system administration commands (invoked from a management station) to the correct administrative API sequence to yield the desired result that is expected upon execution of the system administration command.

The Log Service administrative API functions are applicable to the entities that are controlled by the Log Service such as the Log Stream object.

To date, there are no concurrent and potentially conflicting administrative operations within the scope of the Log Service.

These API functions will be exposed by the IMM Service Object Management library. Only synchronous versions of these API are documented in this version. Support for asynchronous versions will be added later on an as-needed basis based on use cases and requirements.

4.2 Include File and Library Name

The appropriate IMM Service header file and the Log Service header file must be included in the source of an application using the Log Service administration API. For the name of the IMM Service header file, see [3].

4.3 Type Definitions

The specification of Log Service Administration API requires the following types, in addition to the ones already described.

4.3.1 saLogAdminOperationIdT

typedef enum {

SA_LOG_ADMIN_CHANGE_FILTER = 1

} saLogAdminOperationIdT;

4.4 Log Service Administration API

As explained above, the administrative API shall be exposed by the IMM[3] Service library. The IMM Service API *saImmOmAdminOperationInvoke()* or *saImmOmAdminOperationInvokeAsync()* shall be invoked with the appropriate *operationId* (see Section 4.3.1) and *objectName* to execute a particular administrative operation. In the following section, the administrative APIs are described with the assumption that the SA Forum Log Service is an object implementer for the various administrative operations that will be initiated as a consequence of invoking the *saImmOmAdminOperationInvoke()* or the *saImmOmAdminOperationInvokeAsync()* function with the appropriate *operationId* (see Section 4.3.1) on the log stream object designated by *objectName*.

The API syntax for the administrative APIs shall only use the corresponding enumeration value for the *operationId* (see Section 4.3.1) for administrative operations on the Log Service's log stream objects along with *objectName* and the possible return values.

The return values explained in the section below shall be passed in the *operationReturnValue* parameter, which is provided by the invoker of the *saImmOmAdminOperationInvoke()* or the *saImmOmAdminOperationInvokeAsync()* function to obtain return codes from the object implementer (Log Service in this case).

4.4.1 SA_LOG_ADMIN_CHANGE_FILTER

Parameters

operationId -[in] = *SA_LOG_ADMIN_CHANGE_FILTER*

objectName - [in] The LDAP name of the log stream object whose severity filter value is to be changed. The initial RDN type must be "*safLgStr'.* See [4] for SA Forum naming conventions and rules.

param- [in] The severity filter bitmask value to apply to this log stream.

Description

This administrative operation changes the value of the severity filter used on this log stream (see Section 3.3.2.2). The effect is that only log records of the allowed severities are permitted on to the given log stream.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).

SA_AIS_ERR_NOT_EXIST - The logical entity, identified by *objectName,* does not exist in the configuration repository.

SA_AIS_ERR_NOT_SUPPORTED - This administrative procedure is not supported by the type of entity denoted by *objectName*.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has not effect since the provided value is identical to the current value of this log stream severity filter.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is in locked instantiation administrative state.

See Also

SaLogFilterSetCallbackT

5 Alarms and Notifications

The Log Service produces certain alarms and notifications in order to convey important information regarding its operational and functional state to an administrator or a management system.

These reports vary in perceived severity and include alarms, which potentially require an operator intervention and notifications that signify important state or object changes. A management entity should regard notifications, but they do not necessarily require an operator intervention.

The recommended vehicle to be used for producing alarms and notifications is the Notification Service of the Service Availability^{TM} Forum (abbreviated to NTF, see [2]), and hence the various notifications are partitioned into categories as described in this service.

In some cases, this specification uses the word “Unspecified” for values of attributes, which the vendor is at a liberty to set to whatever makes sense in the vendor’s context, and the SA Forum has no specific recommendation regarding such values. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [6])

5.1 Setting Common Attributes

The tables presented in Section 5.2 refer to the attributes in the following list, but do not describe them, as these attributes are described in the list in a generic manner. For each attribute in this list, the specification provides recommendations regarding how to populate the attribute.

- Correlation Ids - They are supplied to correlate two notifications that have been generated because of a related cause. This attribute is optional. But in case of alarms that are generated to clear certain conditions, i.e., produced with a perceived severity of *SA_NTF_SEVERITY_CLEARED*, the correlation id shall be populated by the application with the notification Id that was generated by the Notification Service while invoking the *saNtfNotificationSend()* API during the production of the actual alarm.
- Event Time - The application might pass a timestamp or optionally pass an *SA_TIME_UNKNOWN* value in which case the timestamp is provided by the Notification Service.
- NCI Id - The *vendorId* portion of the *SaNtfClassIdT* data structure must be set to *SA_NTF_VENDOR_ID_SAF* always. The *majorId* and *minorId* will vary based on the specific SA Forum service and the particular notification. Every SA Forum service shall have a *majorId* as described in the enumeration *SaServicesT* (see [4]).

- Notification Id - This attribute is obtained from the Notification Service when a notification is generated, and hence need not be populated by an application.
- Notifying Object - DN of the entity generating the notification. This name must conform to the SA Forum AIS naming convention and contain at least the *safApp* RDN value portion of the DN set to the specified standard RDN value of the SA Forum AIS service generating the notification, which in this case is "*safApp=safLogService*". For details on the SA Forum AIS naming convention, refer to the SA Forum Overview document.

5.2 Log Service Notifications

The following sections describe a set of notifications that a Log Service implementation shall produce.

The value of the *majorId* field within the Notification Class Identifier (*SaNtfClassIdT*) should be set to as follows in all notifications generated by the Log Service.

majorId = SA_SVC_LOG

The *minorId* field within the Notification Class Identifier (*SaNtfClassIdT*) is set distinctly for each individual notification as described below. This field is range-bound, and the used ranges are:

- Alarms: (0x01 - 0x64)
- State change notifications: (0x65 - 0xC8)
- Object change notifications: (0xC9 - 0x12C)
- Attribute change notifications: (0x12D - 0x190)

5.2.1 Log Service Alarms

5.2.1.1 LOG Service Impaired

Description

The Log Service is currently unable to provide service or is in a degraded state because of certain issues with memory, resources, communication or other constraints.

Clearing Method

1) Manual after taking appropriate administrative action or

2) Issue an implementation-specific optional alarm with severity *SA_NTF_SEVERITY_CLEARED* to convey that Log Service self-healed/recovered

and is again providing service. This administrative action is outside of those provided by the Log Service.

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Event Type	Mandatory	*SA_NTF_ALARM_COMMUNICATION*
Notification Object	Mandatory	LOG service, same as Notifying object as specified above.
Notification Class Identifier	NTF internal	*minorId* = 0x01
Additional Text	Optional	“LOG service impaired.”
Additional Information ID	Optional	Unspecified
Probable Cause	Mandatory	Application value from enum *SaNtfProbableCauseT* in [2].
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Application value from enum *SaNtfSeverityT* in [2].
Trend Indication	Optional	Unspecified
Threshold Information	Optional	Unspecified
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

5.2.1.2 Capacity Alarm

Description

This alarm is issued if the ‘log file full action’ is halt and a ‘capacity alarm threshold’ percentage is configured and reached. The particulars of configuring such alarm thresholds is an implementation option to be addressed in some future version of this specification.

Clearing Method

1) Manual after taking appropriate administrative action (such as moving the offending file such that the Log Service automatically creates a new one) or

2) Issue an implementation-specific optional alarm with severity *SA_NTF_SEVERITY_CLEARED* to indicate that the log file is now below the lowest capacity threshold configured.

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Event Type	Mandatory	*SA_NTF_ALARM_PROCESSING*
Notification Object	Mandatory	LOG service, same as Notifying object as specified above.
Notification Class Identifier	NTF internal	*minorId* = 0x02
Additional Text	Optional	“<filename> approaching capacity.”
Additional Information ID	Optional	Unspecified
Probable Cause	Mandatory	Application value from enum *SaNtfProbableCauseT* in [2].
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Application value from enum *SaNtfSeverityT* in [2].
Trend Indication	Optional	*SA_NTF_TREND_MORE_SEVERE* for all alarms after the first alarm.

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Threshold Information	Optional	field values *of SaNtfThresholdInformationT[2] are:* *thresholdId* = *SA_LOG_NTF_LOGFILE_PERCENT_FULL* *thresholdValueType* = *SA_NTF_VALUE_UINT32* *thresholdValue* = *<configured percent value>* *thresholdHysteresis* = *<optional>* *observedValue* = *<observed percent value>*
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

5.2.2 Log Service Object Change Notifications

5.2.2.1 Log Stream Create

Description

This object notification announces the creation of a log stream. It also identifies the location of the log stream’s associated log and configuration files so they can be found and read.

This notification alerts an administrator that log records are now being stored and are available for inspection. It also allows an administrator to be aware that this log stream is operational so that if so desired, the stream's severity bitmask can be adjusted through the *SA_LOG_CHANGE_SEVERITY* administrative operation.

Table 2 Log Stream Create

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Event Type	Mandatory	*SA_NTF_OBJECT_CREATION*
Notification Object	Mandatory	LDAP DN of the log stream created.
Notification Class Identifier	NTF internal	*minorId* = 0xc9.
Additional Text	Optional	"Log stream <log stream name> created"
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION*
Attribute List	Optional	*[0].attributeId = SA_LOG_NTF_ATTR_LOG_STREAM_NAME* *[0].attributeType = SA_NTF_VALUE_STRING* *[0].attributeValue = <stream name>* *[1].attributeId = SA_LOG_NTF_ATTR_LOGFILE_NAME* *[1].attributeType = SA_NTF_VALUE_STRING* *[1].attributeValue = <logfile name>* *[2].attributeId = SA_LOG_NTF_ATTR_LOGFILE_PATH_NAME* *[2].attributeType = SA_NTF_VALUE_STRING* *[2].attributeValue = <path name>*
Attribute Identifier	Optional	Unspecified

5.2.2.2 Log Stream Delete

Description

This object notification announces the deletion of a log stream. It also identifies the location of the log stream's associated log and configuration files so they can be found and read.

This notification alerts an administrator that the log file associated with this log stream is no longer active and perhaps cleanup or archiving chores should commence.

Table 3 Log Stream Delete

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Event Type	Mandatory	*SA_NTF_OBJECT_DELETION*
Notification Object	Mandatory	LDAP DN of the log stream created.
Notification Class Identifier	NTF internal	*minorId* = 0xca.
Additional Text	Optional	"Log stream <log stream name> deleted"
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION*

Table 3 Log Stream Delete

NTF Attribute Name	Parameter Type (X.73Y recommendation or NTF)	SA Forum Recommended value
Attribute List	Optional	*[0].attributeId = SA_LOG_NTF_ATTR_LOG_STREAM_NAME [0].attributeType = SA_NTF_VALUE_STRING [0].attributeValue = <stream name> [1].attributeId = SA_LOG_NTF_ATTR_LOGFILE_NAME [1].attributeType = SA_NTF_VALUE_STRING [1].attributeValue = <logfile name> [2].attributeId = SA_LOG_NTF_ATTR_LOGFILE_PATH_NAME [2].attributeType = SA_NTF_VALUE_STRING [2].attributeValue = <path name>*
Attribute Identifier	Optional	Unspecified