Service Availability™ Forum Application Interface Specification

Log Service SAI-AIS-LOG-A.02.01

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

The Service Availability™ Forum Application Interface Specification (the "Package") found at the URL http://www.saforum.org is generally made available by the Service Availability Forum (the "Copyright Holder") for use in developing products that are compatible with the standards provided in the Specification. The terms and conditions which govern the use of the Package are covered by the Artistic License 2.0 of the Perl Foundation, which is reproduced here.

The Artistic License 2.0

Copyright (c) 2000-2006, The Perl Foundation.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed.

The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.

You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.

Definitions

"Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.

"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.

"You" and "your" means any person who would like to copy, distribute, or modify the Package.

"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.

"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.

"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.

"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.

"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.

"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.

"Source" form means the source code, documentation source, and configuration files for the Package.

"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.

Permission for Use and Modification Without Distribution

(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.

Permissions for Redistribution of the Standard Version

(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.

(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.

Distribution of Modified Versions of the Package as Source

(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:

(a) make the Modified Version available to the Copyright Holder of the Standard Version, under the Original License, so that the Copyright Holder may include your modifications in the Standard Version.

(b) ensure that installation of your Modified Version does not prevent the user installing or running the Standard Version. In addition, the Modified Version must bear a name that is different from the name of the Standard Version.

(c) allow anyone who receives a copy of the Modified Version to make the Source form of the Modified Version available to others under

(i) the Original License or

(ii) a license that permits the licensee to freely copy, modify and redistribute the Modified Version using the same licensing terms that apply to the copy that the licensee received, and requires that the Source form of the Modified Version, and of any works derived from it, be made freely available in that license fees are prohibited but Distributor Fees are allowed.

Distribution of Compiled Forms of the Standard Version or Modified Versions without the Source

(5) You may Distribute Compiled forms of the Standard Version without the Source, provided that you include complete instructions on how to get the Source of the Standard Version. Such instructions must be valid at the time of your distribution. If these instructions, at any time while you are carrying out such distribution, become invalid, you must provide new instructions on demand or cease further distribution.

If you provide valid instructions or cease distribution within thirty days after you become aware that the instructions are invalid, then you do not forfeit any of your rights under this license.

(6) You may Distribute a Modified Version in Compiled form without the Source, provided that you comply with Section 4 with respect to the Source of the Modified Version.

Aggregating or Linking the Package

(7) You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation.

(8) You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package.

Items That are Not Considered Part of a Modified Version

(9) Works (including, but not limited to, modules and scripts) that merely extend or make use of the Package, do not, by themselves, cause the Package to be a Modified Version. In addition, such works are not considered parts of the Package itself, and are not subject to the terms of this license.

General Provisions

(10) Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license.

(11) If your Modified Version has been derived from a Modified Version made by someone other than you, you are nevertheless required to ensure that your Modified Version complies with the requirements of this license.

(12) This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder.

(13) This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed.

(14) Disclaimer of Warranty:

THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

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™ 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™ Forum Application Interface Specification will be used in conjunction with the Service Availability™ Forum Hardware Interface Specification (HPI).

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 ([1]).

1.3 History

The first (and only previous release) of the Log Service specification was:

SAI-AIS-LOG-A.01.01

This section presents the changes of the current release, SAI-AIS-LOG-A.02.01, with respect to the SAI-AIS-LOG-A.01.01 release. Editorial changes are not mentioned here.

1.3.1 New Topics

- Section 3.1.6.1 introduces the *saLogStreamLogFullHaltThreshold* attribute for the halt option of the log file full action.
- Section 3.1.6.4 presents the configuration of the alarm, notification, and system output destination files.
- Section 3.2 describes the behavior of the Log Service API on a cluster node that is not in the cluster membership (see [4]).
- Section 3.4.2.2 introduces new flags for the *SaLogSeverityFlagsT typedef*.
- Section 3.4.6.2 contains the new *SaLogFileCreateAttributesT_2 typedef* that supersedes *SaLogFileCreateAttributesT*. The change was the removal of '*' from

the fields *logFileName*, *logFilePathName*, and *logFileFmt*. See also Section 1.3.3.

- Section 3.4.8 describes the *SaLogLimitIdT* enum, which provides a value that identifies a limit for a particular implementation of the Log Service. The user can inquire at runtime the current value of the limit by specifying the corresponding enum value when invoking the *saLogLimitGet()* function, which is defined in Section Section 3.7.1.
- Section 3.6.1 contains the new functions *saLogStreamOpen_2()* and *saLogStreamOpenAsync_2()* that supersede *saLogStreamOpen()* and *saLogStreamOpenAsync()*. This replacement was necessary because *SaLogFileCreateAttributesT* has been superseded, and the *logStreamName* parameter is now a pointer. See also Section 1.3.3.
- Chapter 4 presents the Log Service UML information model. This model was previously part of [1]. Note that this model has changed compared to the previous version of the Log Service.
- Chapter 5 describes the administration APIs provided for the Log Service.
- Section 6.2.3 introduces the Log Service attribute change notifications.
- Chapter 7 presents the Log Service management interface.

1.3.2 Clarifications

- Section 3.1.5 on the log record output format clarifies that all output is represented using the US-ASCII character set.
- Section 3.1.5.1 clarifies the meaning of some format tokens (@CM, @Cd, @Cy, @CY, @Nm, @Nd, @Ny, @NY, @Cb<fs>, and @Ci<fs>).
- Section 3.1.5.2 clarifies that the @Ci<fs> token can only be used in a format expression that is associated with an application log stream.
- Section 3.1.6 clarifies that the configuration of the alarm, notification, and system log streams can be changed at runtime.
- Section 3.1.6.1 clarifies that the unit of the fixed log record size is byte.
- Section 3.4.6.2 clarifies that the unit of *maxLogRecordSize* is byte.
- Section 3.5.3 on the *saLogDispatch()* function clarifies the meaning of the SA_AIS_OK return value.
- The description of the *saLogFinalize()* function (see Section 3.5.4) clarifies that this function frees all resources allocated by the Log Service for the process in this association between the process and the Log Service.
- The description of the *saLogStreamClose()* function (see Section 3.6.6) clarifies which resources this function frees for the invoking process.
- Section 6.2 clarifies the setting of the notifying object in notifications.

1.3.3 Superseded and Superseding Functions

The Log Service defines for the version A.02.01 two new functions and a new type definition to replace a function and a type definition of the version A.01.01 respectively (see next table).

The superseded functions and type definition are no longer supported in version A.02.01, and no description is provided for them in this document.

Regarding the support of backward compatibility in SA Forum AIS, refer to the Overview document ([1]).

Table 1 Superseded and Superseding Functions in Version A.02.01

Functions or Type Definitions of A.01.01 No Longer Supported in A.02.01	Functions or Type Definitions of A.02.01 Replacing Functions or Type Definitions in A.01.01
saLogStreamOpen()	*saLogStreamOpen_2()*
saLogStreamOpenAsync()	*saLogStreamOpenAsync_2()*
SaLogFileCreateAttributesT	*SaLogFileCreateAttributesT_2*

1.3.4 Changes in Return Values of API Functions:

Table 2 Changes in Return Values of API Functions

API Function	Return Value	Change Type
All API functions except *saLogFinalize()* and *SaLogFilterSetCallbackT*	SA_AIS_ERR_UNAVAILABLE	new
SaLogStreamOpenCallbackT	SA_AIS_ERR_VERSION	new
SaLogStreamOpenCallbackT, *saLogWriteLog()*, *saLogWriteLogAsync()*, and *SaLogWriteLogCallbackT*	SA_AIS_ERR_NO_RESOURCES	extended
SaLogStreamOpenCallbackT	SA_AIS_ERR_INVALID_PARAM	extended
SaLogStreamOpenCallbackT and *SaLogWriteLogCallbackT*	SA_AIS_ERR_BAD_HANDLE SA_AIS_ERR_INVALID_PARAM	new
SaLogStreamOpenCallbackT	SA_AIS_ERR_EXIST	clarified
SA_LOG_ADMIN_CHANGE_FILTER administrative operation function	SA_AIS_ERR_BAD_OPERATION	removed

1.3.5 Removed Topics

SA Forum revisited its alarm issuance directives for this release and modified the conditions that determine when an alarm would be produced. As a consequence, AIS services shall only generate alarms for situations that require an explicit intervention by an external agent or operator, provided that the corrective measures to be taken are well defined. Based on these directives, the alarms generated so far by the AIS services have been revised, and it was decided to remove the "service impaired" alarm from the Log Service A.02.01 version.

SA Forum does not mandate that Log Service implementations which also support the A.01.01 version must generate the "service impaired" alarm for the A.01.01 version.

The "service impaired" alarm has also been removed from the Log Service MIB for the Log Service A.02.01 version.

1.3.6 Other Changes

- Sections 3.1.2.1, 3.1.6, and 3.1.6.3 have changed as the *SaLogStreamConfig* class attributes are now writable.

- The name of the token @Na<fs> for *notificationObject*, which is shown in a row of the table Log Record Format Tokens in Section 3.1.5.1 on page 30, has changed to @No<fs>. As a consequence, the default log record format expression for the application and system log streams shown in Section 3.1.5.3 has changed.
- The definition of the log file path in Section 3.1.6.1 and Section 3.4.6.2 has changed.
- The last attribute in the list contained in Section 3.1.6.1 is new.
- The *notificationId* field in Section 3.4.5.2 must be set to SA_NTF_IDENTIFIER_UNUSED if no identifier is provided. In the previous version may was used instead of must.
- The SA_LOG_ADMIN_CHANGE_FILTER administrative operation was removed from the *SaLogStreamConfig* class (see Section 4.2), as this was a mistake. As a consequence, Section 5.4.1 now states explicitly that this operation applies to the *SaLogStream* runtime UML class and application log streams so represented.
- Section 6.2.1.1 explains under which conditions the capacity alarm is issued. The additional text field has changed.
- Section 6.2.2 has changed because now the Log Service object change notifications only apply to application log streams.

1.4 References

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

[1] Service Availability™ Forum, Service Availability Interface, Overview, SAI-Overview-B.03.01

[2] Service Availability™ Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.02.01

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

[4] Service Availability™ Forum, Application Interface Specification, Cluster Membership Service, SAI-AIS-CLM-B.03.01

[5] Service Availability™ Forum, SA Forum Information Model in XML Metadata Interchange (XMI) v2.1 format, SAI-XMI-A.02.01

[6] Service Availability™ Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.02.01

[7] Service Availability™ Forum, Hardware Platform Interface, SAI-HPI-B.02.01

[8] CCITT Recommendation X.730 | ISO/IEC 10164-1, Object Management Function

[9] CCITT Recommendation X.731 | ISO/IEC 10164-2, State Management Function

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

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

[12] CCITT Recommendation X.736 | ISO/IEC 10164-7, Security Alarm Reporting Function

[13] 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 misconfigurations, 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. An SA Forum Trace Service is on the roadmap, but is not yet defined.

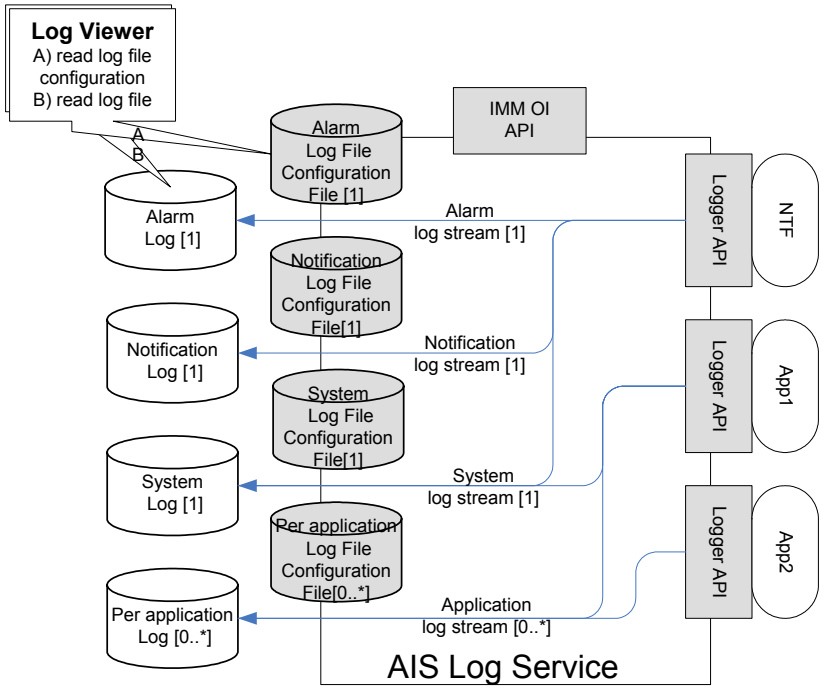
An SA Forum compliant ecosystem assumes the AIS Log Service, or some functionally equivalent service is available for use by applications as well as other AIS services.

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, see [7]) 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 main abstractions of the SA Forum 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 3.6).
- 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 Service (IMM, see [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.

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 3.4.6.1). 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.

Four types of log streams are 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, see [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()* function to introduce a **log record** into a specific log stream (see Section 3.1.2). A logger gains access to a log stream by invoking *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* 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 3.4.6.1). Four distinct types of log streams are supported by the Log Service:

1. **Alarm log stream**: the SA Forum Notification Service (abbreviated to NTF, see [2]) is presumed a client of this Log Service, though this is not mandated. NTF logs alarm information according to the ITU documents alarm reporting (X.733, see [10]) and security alarm reporting (X.736, see [12]). 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 Service (NTF, see [2]) is presumed a client of this Log Service, though this is not mandated. NTF optionally logs notification information according to the ITU documents object management (X.730, see [8]) and state management (X.731, see [9]). 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 noteworthy system circumstances, particularly those that affect service. This log can also be used by AIS services as well as the Availability Management Framework 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 applica-

tion log stream by invoking *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*. Any number of application log streams can exist 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 nor destroyed. Each of these three log streams leads to a stream-specific, mandatory system-defined **log file** or files (see Section 3.4.6.1) and also has an associated log file configuration file (see Section 3.1.6.2).

Log file configuration attributes can be configured by administrative means very early in the lifetime of the cluster or at runtime by using the IMM interface (see [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 invoking *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*. 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 by invoking *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* and by specifying the same log stream and either

- by specifying no other create properties (since the log stream and its properties already exist), or
- by 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.

Any number of private application log streams can exist in a cluster at any given time, each one identified by a cluster-wide unique name. The same application can also have more than one application log stream open at the same time.

An application log stream is destroyed when all application loggers using that stream close it by invoking the *saLogStreamClose()* function. The output destination log file or files (see Section 3.4.6.1) and log file configuration file (see Section 3.1.6.2) associated with the destroyed log stream are closed and persist indefinitely.

If another application log stream is created by invoking *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* and specifying the same log stream name 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 inspecting the log file name changes that have been automatically applied by the Log Service to all completed log files (see Section 3.1.6.2 and Section 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 at which the log is produced). 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 means that only matched log records are allowed entry onto a log stream; all others are discarded. A log filter criterion can only be accessed and configured by administrative means.

Log filtering applies to application and system log records only. 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. Note that the SA Forum Notification Service (see [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 criterion is based on the severity value (see Section 3.4.2.2 on page 42) 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 criterion may qualify that particular nodes, applications, or service units (see [6]) 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)()* function informs a logger about its current filter criteria. This allows a logger to avoid the overhead of packaging and invoking the *saLogWriteLog()* function 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 an *SaLogFilterSetCallbackT* function pointer at *saLogInitialize()* time or did 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 govern the output properties of each log record at an output destination. All output is represented using the US-ASCII character set.

Each format token maps to a specific field or subfield 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, describes 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 lifetime 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

This specification defines 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, 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 token 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 3.4.5.5).

- 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 3 Log Record Format Tokens

Token Type	Description	Example Output Format
@Cr	A 10-digit log record Identifier that the Log Service generates internally. This unsigned 32-bit numeric assignment starts at 1 and increments by 1 as log records arrive at the particular output destination (see Section 3.1.6.5).	‘.......345’
@Ct	18-character hexadecimal representation of time from *logTimeStamp* of type *SaTimeT* in the *SaLogRecordT* structure (see Section 3.4.5.5). 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, 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

Table 3 Log Record Format Tokens (Continued)

Token Type	Description	Example Output Format
@CM	The first three letters (the first one capitalized) of the implied month in English from *logTimeStamp* of type *SaTimeT*.	`Oct`
@Cd	The first three letters (the first one capitalized) of the implied day of the week in English from *logTimeStamp* of type *SaTimeT*.	`Mon`
@Cy	The last two digits of the implied year (since year 2000) from *logTimeStamp* of type *SaTimeT*.	`05`
@CY	The four digits of the implied year from *logTimeStamp* of type *SaTimeT*.	`2005`
@Cc	29-spaced notification class identifier from *notificationClassId* of type *saNtfClassIdT* (see [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 3.4.2.3). If a \0 is found prior to the <fs> length, blank characters will be applied for the remaining characters up to <fs>. This token can be used in format expressions associated with any log stream type. This token can only be used in a format expression associated with an application log streams.	`"port access denied…"` where <fs>=20

Table 3 Log Record Format Tokens (Continued)

Token Type	Description	Example Output Format
@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 3.4.2.3). If the *logBufSize* is less than <fs>, blank characters will be applied for the remaining characters up to <fs>.	"706f72742061636573732064656e6965642020" where <fs>=40; (ascii= "port access denied ")
@Sl<fs>	Logger name from *logSvcUsrName* of type *saNameT* (see Section 3.4.5.3).	'safSu=xx,safSg=yy,safApp=zz...' where <fs>=30
@Sv	2-character severity identifier that maps to one of the SA_LOG_SEV_ severity values (see Section 3.4.2.2). 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* (see [2]), a field in the *SaLogNtfLogHeaderT* structure (see Section 3.4.5.2).	0x0000000000000043
@Nt	18-character hexadecimal representation of time from *eventTime* of type *SaTimeT*, a field in the *SaLogNtfLogHeaderT* structure (see Section 3.4.5.2). Notice that this time is when an alarm or notifications occurred, which is distinct from the time when a log record is logged (see @Ct).	0x0006670634553455

Table 3 Log Record Format Tokens (Continued)

Token Type	Description	Example Output Format
@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, 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`
@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	The first three letters (the first one capitalized) of the implied month in English from *eventTime* of type *SaTimeT*.	`Jan`
@Nd	The first three letters (the first one capitalized) of the implied day of the week in English from *eventTime* of type *SaTimeT*.	`Fri`
@Ny	The last two digits of the implied year (since year 2000) from *eventTime* of type *SaTimeT*.	`11`
@NY	The four digits of the implied year from eventTime of type *SaTimeT*.	`2011`

Table 3 Log Record Format Tokens (Continued)

Token Type	Description	Example Output Format
@Ne<fs>	<*field-size*> hexadecimal expression for event type from type *saNtfEventTypeT* (see [2]), a field in the *SaLogNtfLogHeaderT* structure (see Section 3.4.5.2). 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`).
@No<fs>	*notificationObject* of type *SaNameT*, a field in the *SaLogNtfLogHeaderT* structure (see Section 3.4.5.2).	`'safSu=xx,safSg=yy,safApp=zz.....'` where <fs>=35
@Ng<fs>	*notifyingObject* of type *SaNameT*, a field in the *SaLogNtfLogHeaderT* structure (see Section 3.4.5.2).	`'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, because 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.
   - The @Ci token can only be used in a format expression that is associated with an application log stream, that is, it may not be used in a format expression associated with the three persistent log streams, notification, alarm, and system.
2. If a <*field-size*> is allowed and expressed for a particular token type:

- All character output is left-aligned within its <*field-size*>. If the output is too large, the tail of the character output is truncated.
- All digit or hex output is right-aligned within its <*field-size*>. If the output is too large, 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 underbar ("_"). 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 intention 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, the Log Service will use 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 @No30 @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

Each alarm, notification, or system log stream leads to its 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 by using the IMM service interface (see [3]) available very early in the lifetime of the Log Service and can also be changed at runtime. 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 by invoking the *saLogOpenStream()* function. In this case, the configuration supplied must be correct for the stream to be created (see Section 3.6.1). 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 (see [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 6.2.2), 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 of an application log stream is made operational, the associated file configuration cannot change for the life of that output destination. However, the alarm, notification, and system output destination configurations can be changed at any time during the life of these log streams (see Section 3.1.6.5).

3.1.6.1 Log File Configurable Attributes

The **log file configurable attributes** are:

- **log file path**: this standard relative POSIX pathname specifies the subdirectory (relative to an implementation defined root directory) into which the log file (or files) shall be placed. For each of the notification, alarm, and system log streams, this pathname value cannot be changed (reconfigured). This means all log files created by each of these three log streams shall be located at a known place. The value of the *saLogStreamPathName* attribute for each of the notification, alarm, and system *SaLogStreamConfig* classes explains this location.
- **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**: this attribute specifies the maximum size in bytes to which a log file may grow. Zero means there is no predefined limit. Internally, the Log Service is aware of the current **log file size** (in bytes) so that it is aware when this maximum log file size is reached.
- **fixed log record size**: it indicates the fixed log record size in bytes (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 size 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**: this attribute indicates whether the log file must always be available and implies file replication and persistency. The implementation can achieve replication in any desired fashion (replication, RAID storage, NAS/SAN, etc.) so long as it is accessible from the same pathname from any node in the cluster. Persistency means that the log file must exist across cluster reboots (that is, all nodes go down, then come back, such that 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**: this action specifies the desired Log Service behavior when the maximum log size of a file 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, the *saLogStreamLogFullHaltThreshold* attribute in each of the *SaLogStreamConfig* class instances (for alarm, notification, and system log streams) can be configured to a percent-full threshold value or left alone to assume its default value. Configuration of this threshold value for Application Log Streams is left as an implementation matter. A 'capacity alarm' notification

(see Section 6.2.1.1) shall be generated by the Log Service for both of the two different cases:

- when the log file size is greater than the *saLogStreamLogFullHaltThreshold* value and
- when the log file is now full. No more log records are allowed in this file

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

- **log file format**: explains the log record format expression (see Section 3.1.5.2) that shall be applied to format log records before they are written to the output destination file.

3.1.6.2 Log File Configuration File

When an output destination is specified 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 contains the following 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 and application log record format expression, which is described in Section 3.1.5.3.

When all the log file or files associated with this output destination are closed, and the last log file is closed (for reasons for closure, 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.

Two notable moments exist in the lifetime of a log file (or files) and 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.
- The last application log stream user exits (which is an implicit log stream close).
- A log file has reached maximum capacity and a log file full action (see Section 3.4.6.1) is undertaken, specifically HALT (SA_LOG_FILE_FULL_ACTION_HALT) or ROTATE (SA_LOG_FILE_FULL_ACTION_ROTATE).
- The configuration of the alarm, notification, or system output destination has been changed (see Section 3.1.6.5).

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, several independent output destinations may be 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 if the log file full action is either ROTATE or HALT, and the log file has reached the size given by 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 (SA_LOG_FILE_FULL_ACTION_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 or when this file is associated with the alarm,

notification, or system log stream and the configuration of the file has changed (see Section 3.1.6.5).

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.

Thus, a completed ROTATE log file might read:

```
myLogFile_20050712_102316__20050713_030854.log
```

3.1.6.4 Configuring the Alarm, Notification, and System Output Destination Files

The configuration of the alarm, notification, and system output destination files can be changed while the system is running, as reflected by the writable attributes of the *SaLogStreamConfig* configuration object class (see Section 4.2). However, the consequences of making such configuration changes causes the values in the corresponding <*filename*>.cfg file (see Section 3.1.6.2) to be stale, as this .cfg file no longer reflects the configuration the Log Service is using to format and write log records to the associated output destination log file.

To handle this configuration change, the Log Service shall behave as follows:

- Close the log file for the affected alarm, notification, or system Log Stream and apply the proper log file naming rules for a closed log file (see Section 3.1.6.3).
- Change the name of the corresponding <*filename*>.cfg file (see Section 3.1.6.2) which maintains the association between the now closed .log and this .cfg file.
- Create a new alarm, notification, or system log file and apply the proper log file naming rules (see Section 3.1.6.3).
- Create the associated <*filename*>.cfg file. The content of this file reflects the new output destination file configuration currently used by the Log Service for this log stream's output destination file.

This sequence shall be treated as atomic. That is, log records in and around this configuration change must be properly placed in the old or new .log file according to which configuration applies at the time the log record was formatted and written.

3.1.6.5 Log File Behavior

A log record must be completely written to the log file before a reader is granted read access to that log record. A file reader cannot be blocked from accessing a file to which is currently being written.

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 timestamp). Third party reader tools can use the timestamp 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 Service data type *saNtfClassIdT* (see [2]) provides the principal 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 Unavailability of the Log Service API on a Non-Member Node

The Log Service does not provide service to processes on cluster nodes that are not in the cluster membership (see [4]).

The following subsection describes the behavior of the Log Service under various conditions that cause the Log Service to be unavailable on a node. Section 3.2.2 contains recommendations to Log Service implementers for dealing with a temporary unavailability of providing service.

3.2.1 A Member Node Leaves or Rejoins the Cluster Membership

If the cluster node has left the cluster membership (see [4]) or is being administratively evicted from the cluster membership, the Log Service behaves as follows towards processes residing on that node and using or attempting to use the service:

⇒ Calls to *saLogInitialize()* will fail with SA_AIS_ERR_UNAVAILABLE.

⇒ All Log Service APIs that are invoked by the process and that operate on handles already acquired by the process will fail with SA_AIS_ERR_UNAVAILABLE with the following exceptions:

- Assuming handle *logStreamHandle* has already been acquired: the *saLogWriteLogAsync()* function may return SA_AIS_OK or SA_AIS_ERR_UNAVAILABLE, depending on the service implementation. If it returns SA_AIS_OK, the callback *SaLogWriteLogCallbackT* will be called and will also return SA_AIS_ERR_UNAVAILABLE in the *error* parameter; otherwise, the callback will not be called.
- Assuming handle *logHandle* has already been acquired:
  - The *saLogStreamOpenAsync_2()* function may return SA_AIS_OK or SA_AIS_ERR_UNAVAILABLE, depending on the service implementation. If it returns SA_AIS_OK, the callback *SaLogStreamOpenCallbackT* will be called and will also return SA_AIS_ERR_UNAVAILABLE in the *error* parameter; otherwise, the callback will not be called.
  - The *saLogFinalize()* function is used to free the library handles and all resources associated with these handles.

⇒ Any outstanding callbacks *SaLogStreamOpenCallbackT* and *SaLogWriteLogCallbackT* will return SA_AIS_ERR_UNAVAILABLE in the *error* parameter. The callback *SaLogFilterSetCallbackT* will not be called.

If the node rejoins the cluster membership, processes executing on the node will be able to reinitialize new library handles and use the entire set of Log Service APIs that operate on these new handles; however, invocation of APIs that operate on handles acquired by any process before the node left the membership will continue to fail with SA_AIS_ERR_UNAVAILABLE (or with the special treatment described above for asynchronous calls) with the exception of *saLogFinalize()*, which is used to free the library handles and all resources associated with these handles. Hence, it is recommended for the processes to finalize the library handles as soon as the processes detect that the node left the membership.

When the node leaves the membership, the Log Service executing on the remaining nodes of the cluster behaves as if all processes that were using the Log Service on the leaving node had been terminated. In particular, if the removal of a log stream is

pending because one or more processes on the leaving node had the log stream open, the log stream may be removed now.

3.2.2 Guidelines for Log Service Implementers

The implementation of the Log Service must leverage the SA Forum Cluster Membership Service (see [4]) to determine the membership status of a node for the case explained in Section 3.2.1 before returning SA_AIS_ERR_UNAVAILABLE. If the Cluster Membership Service considers a node as a member of the cluster but the Log Service experiences difficulty in providing service to its clients because of transport, communication, or other issues, it must respond with SA_AIS_ERR_TRY_AGAIN.

3.3 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

3.4 Type Definitions

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

3.4.1 Handles

3.4.1.1 SaLogHandleT

typedef SaUint64T SaLogHandleT;

This type is used for the handle that is supplied by the Log Service to a process during initialization of the Log Service and that is used by the process when it invokes functions of the Log Service API.

3.4.1.2 SaLogStreamHandleT

typedef SaUint64T SaLogStreamHandleT;

This type is used for the handle associated with a particular log stream.

3.4.2 Log Types

3.4.2.1 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 Service (see [2]) to open the alarm log stream, which tracks to the ITU specifications alarm reporting (X.733, see [10]) and security alarm reporting (X.736, see [12]). There is one alarm log stream in a cluster.
- SA_LOG_STREAM_NOTIFICATION - This log stream name is used by the SA Forum Notification Service (see [2]) to open the notification log stream, which tracks to the ITU specifications object management (X.730, see [8]) and state management (X.731, see [9]). 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 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 (historic) circumstances of the cluster. 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. Any number of application log streams can coexist in a cluster.

3.4.2.2 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;


#define SA_LOG_SEV_FLAG_EMERGENCY   0x0001
#define SA_LOG_SEV_FLAG_ALERT       0x0002
#define SA_LOG_SEV_FLAG_CRITICAL    0x0004
#define SA_LOG_SEV_FLAG_ERROR       0x0008
#define SA_LOG_SEV_FLAG_WARNING     0x0010
#define SA_LOG_SEV_FLAG_NOTICE      0x0020
#define SA_LOG_SEV_FLAG_INFO        0x0040

typedef SaUint16T SaLogSeverityFlagsT;
```

The *SaLogSeverityT* type is used to specify the **severity level** of a particular system or application log record (see Section 3.4.5.3) when *saLogWriteLog()* or *saLogWriteLogAsync()* are invoked (see Section 3.6.3).

The *SaLogSeverityFlagsT* type is a bitmap used in the *SaLogFilterSetCallbackT* callback (see Section 3.6.5). 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 and flags have the following interpretation (see [13]):

- EMERGENCY - the system is unusable
- ALERT - action must be taken immediately

- CRITICAL - critical conditions
- ERROR - error conditions
- WARNING - warning conditions
- NOTICE - normal but significant condition
- INFO - informational messages

3.4.2.3 SaLogBufferT

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

A data structure of this type contains the body of the log record and is provided when invoking the *saLogWriteLog()* or the *saLogWriteLogAsync()* functions. The Log Service does not interpret or parse the contents of the area to which *logBuf* points, 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).

3.4.2.4 SaLogAckFlagsT

The *SaLogAckFlagsT* type is used in the *saLogWriteLogAsync()* call. 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.

3.4.2.5 SaLogStreamOpenFlagsT

The following values specify the open attributes used in the *saLogStreamOpen_2()* and *saLogStreamOpenAsync_2()* functions when 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 following value:

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

3.4.3 Log Service API and Notification Types

The Log Service API interface uses the SA Forum Notification Service (see [2]) data types *SaLogNtfLogHeaderT* (see Section 3.4.5.2) and *SaLogGenericLogHeaderT* (see Section 3.4.5.3) as part of their definition. To resolve these data types, the *saLog.h* file simply includes the SA Forum Notification Service (see [2]) header file, as follows:

```
#include <saNtf.h>
```

3.4.4 Log Service as Notification Producer

The Log Service is also a producer of notifications (see Chapter 5). The values placed in certain fields within notifications are assigned by the Log Service.

3.4.4.1 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;
```

3.4.4.2 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         = 2,
        SA_LOG_NTF_ATTR_LOGFILE_PATH_NAME= 3
} SaLogNtfAttributesT;
```

3.4.5 Log Record Types

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

3.4.5.2 SaLogNtfLogHeaderT

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

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

- *notificationId* (defined in *saNtf.h*, see [2]) - This field is a cluster-wide unique identifier value provided to the Log Service by a Notification Service client. This field must be set to SA_NTF_IDENTIFIER_UNUSED (see [2]) if no identifier is provided. The Log Service does not police this value for uniqueness.
- *eventType* (defined in *saNtf.h*, see [2]) - This field reflects the event type of the notification. This field must be set.

- *notificationObject* - A non-NULL pointer to the name of the logical entity (identified by its full LDAP name) about which the notification is generated.
- *notifyingObject* - A non-NULL pointer to the name of the logical entity (identified by its full LDAP name) that is sending the notification. This field must be set.
- *notificationClassId* - A pointer to an *SaNtfClassIdT* structure (defined in *saNtf.h*, see [2]) that uniquely identifies the kind of situation that caused the notification. This field is optional.
- *eventTime* - This field contains the time at which an event is detected. This time may not be the same time at which the event was reported or the notification was logged. This field must be set.

3.4.5.3 SaLogGenericLogHeaderT

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

A structure of this type 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*, see [2]) - A pointer to a structure of type *SaNtfClassIdT* which is used for internationalization. This field is optional and 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* - A pointer to the LDAP name used by the logger to identify itself. This name will typically identify a component or service unit, provided the user is a component under the control of the Availability Management Framework (see [6]). 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 when invoking *saLogWriteLog()* or *saLogWriteLogAsync()*, 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 3.4.2.2.

3.4.5.4 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, the *ntfHdr* structure must be properly populated (refer to Section 3.4.5.2). If the log record is destined for either the system or an application log stream, *genericHdr* must be properly populated (refer to Section 3.4.5.3).

3.4.5.5 SaLogRecordT

The following type 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 timestamp cannot be provided by the user, the constant SA_TIME_UNKNOWN shall be specified instead, which means the Log Service needs to supply the timestamp.
- *logHdrType* - This field must be set. It indicates the log record header type that is populated in the *SaLogHeaderT* union (see Section 3.4.5.4) of the next parameter, *logHeader*.
- *logHeader* - For details on how to populate this field based on the *logHdrType* field, refer to Section 3.4.5.4.
- *logBuffer* - This field is a pointer to a buffer containing the body of the log record, which the Log Service treats as a single opaque data unit. The *logBuffer* pointer 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.

3.4.6 Application Log Types

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

3.4.6.1 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 specifies the Log Service behavior when the maximum log size of a file is reached. This policy is specified when 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.

3.4.6.2 SaLogFileCreateAttributesT_2

```
typedef struct {
    SaStringT           logFileName;
    SaStringT           logFilePathName;
    SaUint64T           maxLogFileSize;
    SaUint32T           maxLogRecordSize;
    SaBoolT             haProperty;
    SaLogFileFullActionT logFileFullAction;
    SaUint16T           maxFilesRotated;
    SaStringT           logFileFmt;
} SaLogFileCreateAttributesT_2;
```

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

- *logFileName* - A pointer to the POSIX log file name to be associated with an application specific log stream. A value must be set.
- *logFilePathName* - A pointer to a relative POSIX pathname that qualifies the subdirectory (relative to an implementation defined directory) where the log file resides. References to the parent directory ("..") cannot be included in *logFilePathName*. Subdirectories included in *logFilePathName* will be automatically created by the Log Service. A NULL pointer is equivalent to the string with a single dot ("."), which refers to the implementation defined directory itself.
- *maxLogFileSize* - The maximum size in bytes to which a log file may grow. A value of zero indicates no predefined limit. If the specified limit is exceeded, the *logFileFullAction* action (see below) is invoked. A value must be set.
- *maxLogRecordSize* - The maximum log record size in bytes 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. For details, refer to Section 3.4.6.1. 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* - A pointer to a memory area containing a log record format expression specified by the logger. If this field is NULL, the Log Service uses the default format expression for the target log stream type (see Section 3.1.5.3).

3.4.7 SaLogCallbacksT

The *SaLogCallbacksT* structure is defined as follows:

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

A structure of the *SaLogCallbacksT* type (called a callbacks structure) is used to specify the callback functions that the Log Service will invoke at well-defined moments.

3.4.8 SaLogLimitIdT

The *SaLogLimitIdT* enum provides a value that identifies a limit for a given implementation of the Log Service. Note that the Log Service specification does not define a configuration for this limit, which is usually predefined by the implementation.

The user can retrieve at runtime the current value of this limit by specifying the identifier of the limit (the enum value of the type *SaLogLimitIdT*, defined below) when invoking the *saLogLimitGet()* function (see Section 3.7.1 on page 75).

The limit value is returned in a parameter of a generic type (*SaLimitValueT* type, defined in [1]). As the only limit defined in this specification is of type *SaUint64T*, the *uint64Value* field of *SaLimitValueT* must be used for further access.

```
typedef enum {
    SA_LOG_MAX_NUM_CLUSTER_APP_LOG_STREAMS_ID = 1
} SaLogLimitIdT;
```

The only value of the *SaLogLimitIdT* enumeration type has the following interpretation:

- SA_LOG_MAX_NUM_CLUSTER_APP_LOG_STREAMS_ID - This enum can be used to retrieve the maximum number of cluster-wide application log streams that may be created in the cluster.

3.5 Library Life Cycle

3.5.1 saLogInitialize()

Prototype

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

Parameters

logHandle - [*out*] A pointer to the handle which designates this particular initialization of the Log Service, and which is to be returned by the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

logCallbacks - [*in*] If *logCallbacks* is set to NULL, no callback is registered; If *logCallbacks* is not set to NULL, it is a pointer to an *SaLogCallbacksT* structure which contains the callback functions of the process that the Log Service may invoke. Only non-NULL callback functions in this structure will be registered. The *SaLogCallbacksT* type is defined in Section 3.4.7 on page 51.

version - [*in*/*out*] As an input parameter, *version* is a pointer to a structure containing the required Log Service version. In this case, *minorVersion* is ignored and should be set to 0x00.
As an output parameter, *version* is a pointer to a structure containing the version actually supported by the Log Service. The *SaVersionT* type is defined in [1].

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 pointed to by *logHandle* is returned by the Log Service 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 specified *releaseCode* and *majorVersion*, SA_AIS_OK is returned. In this case, the structure pointed to by 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 preceding condition cannot be met, SA_AIS_ERR_VERSION is returned, and the structure pointed to by 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 lowest 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 lower 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 provided in the structure to which the *version* parameter points is not compatible with the version of the Log Service implementation.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogSelectionObjectGet(), *saLogDispatch()*, *saLogFinalize()*

3.5.2 saLogSelectionObjectGet()

Prototype

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

Parameters

logHandle - [*in*] The handle which was obtained by a previous invocation of the *saLogInitialize()* function and which designates this particular initialization of the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

selectionObject - [*out*] A pointer to the operating system handle that the process can use to detect pending callbacks. The *SaSelectionObjectT* type is defined in [1].

Description

This function returns the operating system handle, *selectionObject*, associated with the handle *logHandle*. The invoking process can use the operating system 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 operating system handle 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).

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogInitialize(), *saLogDispatch()*, *saLogFinalize()*

3.5.3 saLogDispatch()

Prototype

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

Parameters

logHandle - [*in*] The handle which was obtained by a previous invocation of the *saLogInitialize()* function and which designates this particular initialization of the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

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. These flags are values of the *SaDispatchFlagsT* enumeration type, which is described in [1].

Description

In the context of the calling thread, this function invokes pending callbacks for the handle *logHandle* in the way specified by the *dispatchFlags* parameter.

Return Values

SA_AIS_OK - The function completed successfully. This value is also returned if this function is being invoked with *dispatchFlags* set to SA_DISPATCH_ALL or SA_DISPATCH_BLOCKING, and the handle *logHandle* has been finalized.

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.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogInitialize(), *saLogFinalize()*

3.5.4 saLogFinalize()

Prototype

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

Parameters

logHandle - [*in*] The handle which was obtained by a previous invocation of the *saLogInitialize()* function and which designates this particular initialization of the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

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 completes successfully, it 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 *saLogStreamOpenCallbackT* callbacks related to the particular handle. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

If a process terminates, the Log Service implicitly finalizes all instances of the Log Service that are associated with the process, as described in the preceding paragraph.

After *saLogFinalize()* completes successfully, the handle *logHandle* and the selection object associated with it are no longer valid.

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(), *saLogStreamClose()*, *saLogSelectionObjectGet()*, *saLogStreamOpenCallbackT*

3.6 Log Service Operations

3.6.1 saLogStreamOpen_2() and saLogStreamOpenAsync_2()

Prototype

```
SaAisErrorT saLogStreamOpen_2(
    SaLogHandleT                        logHandle,
    const SaNameT                       *logStreamName,
    const SaLogFileCreateAttributesT_2  *logFileCreateAttributes,
    SaLogStreamOpenFlagsT               logStreamOpenFlags,
    SaTimeT                             timeout,
    SaLogStreamHandleT                  *logStreamHandle
);

SaAisErrorT saLogStreamOpenAsync_2(
    SaLogHandleT                        logHandle,
    const SaNameT                       *logStreamName,
    const SaLogFileCreateAttributesT_2  *logFileCreateAttributes,
    SaLogStreamOpenFlagsT               logStreamOpenFlags,
    SaInvocationT                       invocation
);
```

Parameters

logHandle - [*in*] The handle which was obtained by a previous invocation of the *saLogInitialize()* function and which designates this particular initialization of the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

logStreamName - [*in*] This parameter points to the DN name of the log stream to open. This name may be one of the well-known log stream names (see Section 3.4.2.1), or it may be a user-defined cluster-wide unique application log stream name. The *SaNameT* type is defined in [1].

logFileCreateAttributes - [*in*] A pointer to the *SaLogFileCreateAttributesT_2* structure (as defined in Section 3.4.6.2 on page 50) that describes the attributes associated

with an application log stream only. If one of the well-known log streams is being opened, this pointer must be NULL. Other considerations are as follows:

- If the user intends only to open an existing application log stream by supplying the same log stream name, this value must be NULL.
- If the user intends to open and create an application log stream that does not yet exist, an *SaLogFileCreateAttributesT_2* structure must be populated and its pointer passed in *logFileCreateAttributes*.
- If the user intends to open a (possibly) existing application log stream, but still specify creation attribute values, the provided values must be identical to those values provided by the initial logger that successfully created the application log stream.

logStreamOpenFlags - [*in*] The value of this parameter is either zero or SA_LOG_STREAM_CREATE, as defined in the *SaLogStreamOpenFlagsT* type in Section 3.4.2.5 on page 43. The SA_LOG_STREAM_CREATE value may only be set when opening an application log stream. If one of the well-known log streams is being opened, this value must not be set. Other considerations are as follows:

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

timeout - [*in*] The *saLogStreamOpen_2()* 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. The *SaTimeT* type is defined in [1].

invocation - [*in*] This parameter allows the invoking logger to match this invocation of *saLogStreamOpenAsync_2()* with the corresponding *(*SaLogStreamOpenCallbackT)()* callback call. The *SaInvocationT* type is defined in [1].

logStreamHandle - [*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 the memory area to which *logStreamHandle* points the handle that the logger uses to access the correct log stream in subsequent invocations of the functions of the Log Service Operations APIs. The *SaLogStreamHandleT* type is defined in Section 3.4.1.2 on page 40.

Description

The *saLogStreamOpen_2()* and *saLogStreamOpenAsync_2()* functions open a log stream. If the log stream is an application log stream and the named application log stream does not exist, the structure to which *logFileCreateAttributes* points must be populated and the SA_LOG_STREAM_CREATE flag must be set in the *logStreamOpenFlags* parameter.

For the three well-known log streams, the returned log stream handle refers to 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 lifetime of the Log Service in the cluster.

An invocation of *saLogStreamOpen_2()* 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_2()* 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_2()* 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 wanted, a different format may be specified when creating the log stream by using the syntax described in Section 3.1.5. After a format expression is associated with a log stream, it cannot be changed over the lifetime 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 invocation of *saLogInitialize()* to initialize the Log Service was incomplete, since the *saLogStreamOpenCallbackT* callback function is missing. This applies only to the *saLogStreamOpenAsync_2()* 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.
- One of the well-known log stream names is identified and one or both of the following cases occur:
  - The SA_LOG_STREAM_CREATE flag is set.
  - The *logFileCreateAttributes* is not NULL.
- An application log stream is identified, and a format expression has been provided, but the format expression is not well formed (see Section 3.1.5.2).
- The *logStreamName* parameter does not point to a DN, or the type of its first RDN 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). In particular, this value is returned if the maximum number of application log streams allowed has been reached, as reflected by the SA_LOG_MAX_NUM_CLUSTER_APP_LOG_STREAMS_ID enum (see Section 3.4.8).

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

SA_AIS_ERR_EXIST - The SA_LOG_STREAM_CREATE flag is set in *logStreamOpenFlags*, the application log stream designated by the name pointed to by *logStreamName* already exists, and the values in the structure to which *logFileCreateAttributes* points do not match the values used to originally open this application log stream.

SA_AIS_ERR_BAD_FLAGS - The *logStreamOpenFlags* parameter is invalid.

SA_AIS_ERR_VERSION - The invoked function is not supported in the version specified in the call to initialize this instance of the Log Service library.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogStreamClose(), *saLogStreamOpenCallbackT*, *saLogInitialize()*

3.6.2 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_2()* 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. The *SaInvocationT* type is defined in [1].

logStreamHandle - [*in*] The handle that designates the log stream if *error* is SA_AIS_OK. The *SaLogStreamHandleT* type is defined in Section 3.4.1.2 on page 40.

error - [*in*] This parameter indicates whether the *saLogStreamOpenAsync_2()* function was successful. The *SaAisErrorT* type is defined in [1]. 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_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. 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.
  - One of the well-known log stream names is identified and one or both of the following cases occur:
    - The SA_LOG_STREAM_CREATE flag is set.
    - The *logFileCreateAttributes* is not NULL.
  - An application log stream is identified, and a format expression has been provided, but the format expression is not well formed (see Section 3.1.5.2).
  - The *logStreamName* parameter does not point to a DN, or the type of its first RDN is not *safLgStr*.
- 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). In particular, this value is returned if the maximum number of application log streams allowed has been reached, as reflected by the SA_LOG_MAX_NUM_CLUSTER_APP_LOG_STREAMS_ID enum (see Section 3.4.8).
- SA_AIS_ERR_NOT_EXIST - The SA_LOG_STREAM_CREATE flag is not set, the *logFileCreateAttributes* is NULL, and the application log stream designated by the name pointed to by *logStreamName* does not exist.
- SA_AIS_ERR_EXIST - The SA_LOG_STREAM_CREATE flag is set in *logStreamOpenFlags*, the application log stream designated by the name pointed to by *logStreamName* already exists, and the values in the structure to which *logFileCreateAttributes* points do not match the values used to originally open this application log stream.
- SA_AIS_ERR_BAD_FLAGS - The *logStreamOpenFlags* parameter is invalid.
- SA_AIS_ERR_VERSION - The invoked function is not supported in the version specified in the call to initialize this instance of the Log Service library.

- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

Description

The Log Service calls this callback function when the operation requested by the invocation of *saLogStreamOpenAsync_2()* completes.

This callback is invoked in the context of a thread calling *saLogDispatch()* on the handle *logHandle* that was specified in the *saLogStreamOpenAsync_2()* call.

If this call completes successfully, the reference to the opened or created and opened log stream is returned in *logStreamHandle*; otherwise, an error is returned in the error parameter.

Return Values

None

See Also

saLogStreamOpenAsync_2(), *saLogDispatch()*, *saLogInitialize()*

3.6.3 saLogWriteLog() and saLogWriteLogAsync()

Prototype

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

SaAisErrorT saLogWriteLogAsync(
        SaLogStreamHandleT      logStreamHandle,
        SaInvocationT           invocation,
        SaLogAckFlagsT          ackFlags,
        const 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 *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*. The *SaLogStreamHandleT* type is defined in Section 3.4.1.2 on page 40.

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. The *SaTimeT* type is defined in [1].

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. The *SaLogAckFlagsT* type is defined in Section 3.4.2.4 on page 43.

logRecord - [*in*] A non-NULL pointer to a structure of type *SaLogRecordT* describing the contents of the log record. The various fields of this type are described in Section 3.4.5.5 on page 48, which gives 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. The *SaInvocationT* type is defined in [1].

Description

These API functions are used to log a record to which *logRecord* points 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*.

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

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 with the *saLogStreamOpen_2()* or *saLogStreamOpenCallback()* functions, or the corresponding log stream has already been closed.
- The handle *logHandle* that was passed to the *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* functions has already been finalized.

SA_AIS_ERR_INIT - The previous invocation of *saLogInitialize()* to initialize the Log Service 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 the *SaLogRecordT* structure to which *logRecord* points does not correspond to the type of log stream implied by *logStreamHandle*.
- The *logSvcUsrName* (see Section 3.4.5.3) 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). In particular, this value is returned if the output 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.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

SaLogWriteLogCallbackT, *saLogStreamOpen_2()*, *saLogStreamOpenAsync_2()*, *saLogStreamOpenCallbackT*, *saLogInitialize()*

3.6.4 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. The *SaInvocationT* type is defined in [1].

error - [*in*] This parameter indicates whether the *saLogWriteLogAsync()* function was successful. The *SaAisErrorT* type is defined in [1]. 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_BAD_HANDLE - The handle *logStreamHandle* is invalid, due to one or both of the reasons below:
  - It is corrupted, was not obtained with the *saLogStreamOpen_2()* or *saLogStreamOpenCallback()* functions, or the corresponding log stream has already been closed.
  - The handle *logHandle* that was passed to the *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* functions has already been finalized.
- 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 the *SaLogRecordT* structure to which *logRecord* points does not correspond to the type of log stream implied by *logStreamHandle*.

- The *logSvcUsrName* (see Section 3.4.5.3) 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). In particular, this value is returned if the output 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.
- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

Description

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

This callback is invoked in the context of a thread calling *saLogDispatch()* on the handle *logHandle* that was specified in the corresponding *saLogWriteLogAsync()* call.

If successful, the log record is written to the destination log file associated with the log stream designated by *logStreamHandle* in the invocation of the corresponding *saLogWriteLogAsync()* function.

Return Values

None

See Also

saLogWriteLogAsync(), *saLogDispatch()*, *saLogInitialize()*

3.6.5 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 must have been obtained previously by the invocation of *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*. The *SaLogStreamHandleT* type is defined in Section 3.4.1.2 on page 40.

logSeverity - [*in*] This parameter specifies which log records are allowed to be forwarded from a logger source. It is a bitmap that describes the severity levels at which logging is enabled, that is, only log records with severity levels enabled in the *logSeverity* bitmap will be forwarded to the Log Service. The *SaLogSeverityFlagsT* type is defined in Section 3.4.2.2 on page 42.

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.

After this function is invoked, 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()* or *saLogWriteLogAsync()* and will discard any log records whose severity level is disabled.

This callback may be invoked as a consequence of an administrative operation to set a particular log stream at desired severity levels or as a matter of initial configuration, which causes a preconfigured *logSeverity* to be pushed to the affected processes that are linked with the Log Service library. This callback can be invoked at any time

after a successful completion of *saLogStreamOpen_2()* or of the *(*SaLogStreamOpenCallbackT)()* callback.

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

Return Values

None

See Also

saLogWriteLog(), *saLogWriteLogAsync()*, *saLogStreamOpen_2()*, *saLogStreamOpenAsync_2()*, *SaLogFilterSetCallbackT*

3.6.6 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 *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*. The *SaLogStreamHandleT* type is defined in Section 3.4.1.2 on page 40.

Description

The invocation of this API function closes the log stream which is designated by *logStreamHandle* and which was opened by an earlier invocation of *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()*.

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

This call frees all resources allocated for this process by the Log Service on the log stream identified by the handle *logStreamHandle*.

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.

If the invocation of the *saLogStreamClose()* function completes successfully, and the log stream is an application log stream, and no other process has that application log stream open, the Log Service behaves as follows.

- The log stream is deleted.
- 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 Section 3.1.6.5).
- The log file configuration file (see Section 3.1.6.2) associated with the deleted log stream is closed and persists indefinitely.

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 with the *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* functions, or the corresponding log stream has already been closed.
- The handle *logHandle* that was passed to the *saLogStreamOpen_2()* or *saLogStreamOpenAsync_2()* functions has already been finalized.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogStreamOpen_2(), *saLogStreamOpenAsync_2()*, *SaLogWriteLogCallbackT*, *SaLogFilterSetCallbackT*

3.7 Limit Fetch API

3.7.1 saLogLimitGet()

Prototype

SaAisErrorT saLogLimitGet(

SaLogHandleT logHandle,

SaLogLimitIdT limitId,

*SaLimitValueT *limitValue*

);

Parameters

logHandle - [*in*] The handle which was obtained by a previous invocation of the *saLogInitialize()* function and which designates this particular initialization of the Log Service. The *SaLogHandleT* type is defined in Section 3.4.1.1 on page 40.

limitId - [*in*] The Log Service limit whose implementation-specific value needs to be queried. The limits are defined in the *SaLogLimitIdT* type in Section 3.4.8 on page 52.

limitValue - [*out*] Pointer to the actual value of the limit specified in *limitId*. For details regarding this type, refer to the SA Forum Overview document ([1]).

Description

This function enables a user application to obtain the current implementation-specific value of a Log Service limit. The *limitId* parameter represents the limit to be queried. When this function completes successfully, it returns the current value of the specified limit in the memory area pointed to by *limitValue*.

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. This error is returned due to one or both of the following reasons:

- The *limitId* parameter contains an invalid value.
- The *limitValue* pointer is NULL.

SA_AIS_ERR_VERSION - The invoked function is not supported in the version specified in the call to initialize this instance of the Log Service library.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saLogInitialize()

4 Log Service UML Information Model

The Log Service information model is described in UML and has been organized in a UML class diagram.

The Log Service UML model is implemented by the SA Forum IMM Service [3]. For details on this implementation, refer to the SA Forum Overview document ([1]).

The Log Service UML class diagram has two classes, which show the contained attributes and the administrative operations (if any) applicable on these classes.

4.1 DN Format for Log Service UML Classes

Table 4 DN Formats for Objects of Log Service Classes

Object Class	DN Formats for Objects of the Class
SaLogStream	"*safLgStr*=…,"
SaLogStreamConfig	"*safLgStrCfg*=…,"

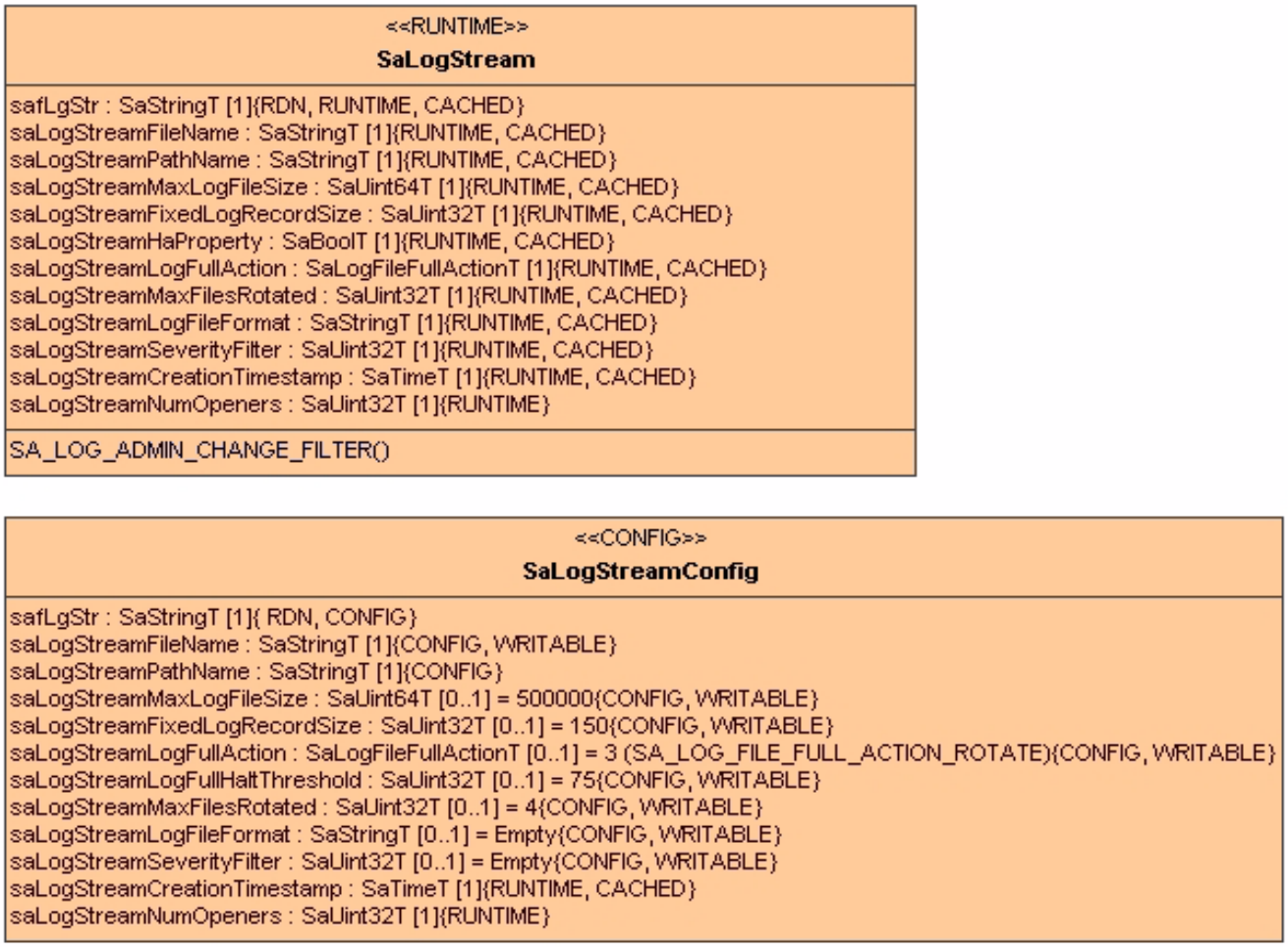
4.2 Log Service UML Classes

The two classes of the Log Service UML model are:

- *SaLogStream*—This is a runtime object class that exposes various runtime attributes associated with each application log stream in the cluster.
- *SaLogStreamConfig*—This is a configuration object class. There are always three and only three of these classes in a cluster, which correspond to the alarm, notification, and system log streams. This class allows the administrator to change configuration properties for any of these three log streams in a running system.

FIGURE 2 shows these classes. A description of each attribute of these classes may be found in the XMI file (see [5]). For additional details, refer to the SA Forum Overview document ([1]).

FIGURE 2 Log Service UML Classes



5 Log Service Administration API

5.1 Log Service Administration API Model

5.1.1 Log Service Administration API Basics

This section describes the 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 such as SNMP agents 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.

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.

5.2 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].

5.3 Type Definitions

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

5.3.1 saLogAdminOperationIdT

typedef enum {

SA_LOG_ADMIN_CHANGE_FILTER = 1

} saLogAdminOperationIdT;

5.4 Log Service Administration API

As explained above, the administrative API shall be exposed by the IMM (see [3]) Service library. The IMM Service API *saImmOmAdminOperationInvoke()* or *saImmOmAdminOperationInvokeAsync()* shall be invoked with the appropriate *operationId* (see Section 5.3.1) 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 administrative operations that will be initiated as a consequence of invoking the *saImmOmAdminOperationInvoke()* or the *saImmOmAdminOperationInvokeAsync()* functions with the appropriate *operationId* (see Section 5.3.1) on the log stream object designated by the name to which *objectName* points.

The API syntax for the administrative APIs shall use only the corresponding enumeration value for the *operationId* (see Section 5.3.1) for administrative operations on Log Service's log stream objects designated by the name to which *objectName* points, leading to the possible return values.

The return values explained in the section below shall be passed in the *operationReturnValue* parameter that 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).

5.4.1 SA_LOG_ADMIN_CHANGE_FILTER

Parameters

operationId - [*in*] = SA_LOG_ADMIN_CHANGE_FILTER

objectName - [*in*] Pointer to the LDAP name of the application log stream object whose severity filter value is to be changed. The initial RDN type must be "*safLgStr*". For the SA Forum naming conventions and rules, see [1].

params - [*in*] NULL-terminated array of pointers to parameter descriptors. The parameter descriptor corresponds here to the severity filter bitmask value to apply to this log stream.

Description

This administrative operation is only valid on *SaLogStream* runtime objects which only ever represent application log streams.

This administrative operation changes the value of the severity filter used on this application log stream (see Section 3.4.2.2). 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 the name to which *objectName* points 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 the name to which *objectName* points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect since the provided value is identical to the current value of this log stream severity filter.

See Also

SaLogFilterSetCallbackT

6 Alarms and Notifications

The Log Service produces certain alarms and notifications to convey important information regarding

- its own operational and functional state and
- the operational and functional state of the objects under its control

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™ 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 term “Unspecified” for values of attributes. This means that the SA Forum has no specific recommendation on the setting, and the vendor may set these attributes to whatever makes sense in the vendor’s context. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [10]).

6.1 Setting Common Attributes

The tables presented in Section 6.2 refer to attributes defined in [2]. The following list provides recommendations regarding how to populate these attributes.

- Correlation ids - They are supplied to correlate two notifications that have been generated because of a related cause. This attribute is optional; however, in case of alarms that are generated to clear certain conditions, that is, 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 when 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 notification class identifier is an attribute of type *SaNtfClassIdT*. 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 [1]).
- 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 ([1]).

6.2 Log Service Notifications

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

The notifying object must be set to the DN "*safApp=safLogService*" for the Log Service.

The value of the *majorId* field in the notification class identifier (*SaNtfClassIdT*) must be set to SA_SVC_LOG (as defined in the *SaServicesT* enum in [1]) in all notifications generated by the Log Service.

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)

6.2.1 Log Service Alarms

6.2.1.1 Capacity Alarm

Description

This alarm is issued for both of these two different cases:

- when the 'log file full action' is halt, and the log file size is greater than the *saLogStreamLogFullHaltThreshold* value (see Section 3.1.6.1) and
- when the log file is now full. No more log records are allowed in this file.

The capacity alarm threshold can be configured for the alarm, notification, and system log streams by default or by setting the *saLogStreamLogFullHaltThreshold* attribute in the corresponding *SaLogStreamConfig* configuration object class instance when the attribute *saLogStreamLogFullAction* is set to 'halt'. The configuration of the application log stream alarm threshold value and its default value are currently implementation-specific.

Clearing Method

1) Manual after taking appropriate administrative action or

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

Table 5 Log Service Capacity Alarm

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_ALARM_PROCESSING
Notification Object	Mandatory	"*safApp=safLogService*"
Notification Class Identifier	NTF-internal	*minorId* = 0x02
Additional Text	Optional	"<*filename*> has reached full capacity."
Additional Information	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.
Threshold Information	Optional	Field values of *SaNtfThresholdInformationT*, as defined in [2]: *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

6.2.2 Log Service Object Change Notifications

6.2.2.1 Application Log Stream Create

Description

This object change notification announces the creation of an application log stream. It also identifies the location of the application 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 application log stream is operational so that, if so desired, the stream's severity bitmask can be adjusted with the SA_LOG_CHANGE_SEVERITY administrative operation.

.

Table 6 Application Log Stream Creation Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_CREATION
Notification Object	Mandatory	LDAP DN of the application log stream (as specified in Section 4.1) that has been created.
Notification Class Identifier	NTF-internal	*minorId* = 0xc9.
Additional Text	Optional	“Application log stream <log stream name> created”
Additional Information	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* = <pathname>

6.2.2.2 Application Log Stream Delete

Description

This object change notification announces the deletion of an application log stream. It also identifies the location of the application 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 application log stream is no longer active and perhaps cleanup or archiving chores should commence.

Table 7 Application Log Stream Deletion Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_DELETION
Notification Object	Mandatory	LDAP DN of the application log stream (as specified in Section 4.1) that has been created.
Notification Class Identifier	NTF-internal	*minorId* = 0xca.
Additional Text	Optional	“Application log stream <log stream name> deleted”
Additional Information	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* = <pathname>

6.2.3 Log Service Attribute Change Notifications

6.2.3.1 Log Stream Attribute Change

Description

This notification is generated when one or more class attributes associated with a *SaLogStreamConfig* class has changed. There is one *SaLogStreamConfig* class for each of three persistent log streams: notification, alarm, and system.

The consequences of a configuration change is that the Log Service shall close the current <*filename*>.cfg file (and rename it to <*filename*>_<*closetime*>.cfg) and the current <*filename*>_<*opentime*>.log file (and rename it to <*filename*>_<*createtime*>_<*closetime*>.log) and open up new .cfg and .log files as formally described in Section 3.1.6.2 and in Section 3.1.6.3.

This notification alerts an administrator that these file changes have occurred and that the now closed file or files are available for compacting, archiving, or any other implementation specific action.

Only the ‘old’ (now closed) log file name and the ‘new’ (now open) log file name attribute change is actually reported. This is enough information for an administrator to find and manage the relevant files, given:

- the log file configuration file naming rules (see Section 3.1.6.2) and the log file naming rules (see Section 3.1.6.3), and
- the log file pathname attribute for each of the three persistent log streams cannot change (see Section 3.1.6.1) so the location of the relevant files is known.

Table 8 Log Stream Attribute Change

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_ATTRIBUTE_CHANGED
Notification Object	Mandatory	LDAP DN of one of the persistent log streams.
Notification Class Identifier	NTF-internal	*minorId* = 0x12d.
Additional Text	Optional	Filename for log stream <log stream name> has changed.
Additional Information	Optional	Unspecified
Source Indicator	Mandatory	SA_NTF_OBJECT_OPERATION
Changed Attribute List	Mandatory	[0].*attributeId* = SA_LOG_NTF_ATTR_LOGFILE_NAME [0].*attributeType* = SA_NTF_VALUE_STRING [0].*oldAttributePresent* = SA_TRUE [0].*oldAttributeValue* = <*filename*>_<*createtime*>_<*closetime*>.log [0].*newAttributeValue* = <*filename*>_<*createtime*>.log

7 Log Service Management Interface

Currently, an SNMP MIB interface is defined for the Log Service. Other management access methods to the Log Service may be added in future versions of this specification.

7.1 Log Service MIB (SAF-LOG-SVC-MIB)

The Log Service MIB contains the single read-only table *saLogStreamTable*, which enumerates the attributes of all log streams that currently exist in the cluster. This includes the required and persistent alarm, notification, and system log streams as well as any number of application log streams that currently exist. This table mimics the UML object classes *SaLogStream* and *SaLogStreamConfig*, as described in Section 4.2 in terms of the objects contained in the table.

Additionally, the Log Service MIB also defines SNMP traps that correspond to the various notifications for the service defined in Chapter 6 of this specification.

For a detailed description of the various objects of this MIB, refer to the SAF-LOG-SVC-MIB as can be downloaded from http://www.saforum.org/specification/download/get_spec.

Index of Definitions