

Java Usage of the SAF Log Service



Open Specifications for Service Availability

Service availability is defined as the delivery of uninterrupted services despite any system component failure.

A system configured with redundant components that fail could switch over to the standby components without jeopardizing the metrics of reliability or availability. Service availability encompasses both the requirements for high availability and the requirements for high reliability as they pertain to continuity of service,

Contents

1	Introduction
2	Architecture
3	Integration Requirements
4	Application Requirements
5	Use cases

1 Introduction

1.1 Purpose

Service Availability Forum has specified interfaces for highly available software. Application Interface Specification (AIS) defines the interface between applications and the middleware providing service availability within a cluster to enable a system with no single point of failure.

AIS has been defined using the C language conventions. The purpose of this document is to define how Java applications can use the SAF Log Service.

1.2 Concepts

- **Integration Implementation**
The implementation that integrates between SAF and Java. This part uses the APIs defined by SAF and maps between standard Java APIs and SAF APIs. The integration implementation does not contain any user defined code.
- **Application**
An application in user defined Java code. The Application does not include generic Java code of the Java runtime environment or of the Integration Implementation. Nor does it include the code of application servers on which Java EE applications are deployed.

1.3 References

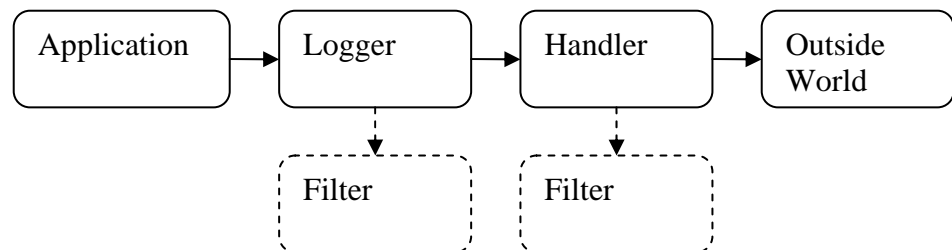
- [1] Service Availability Forum, Application Interface Specification, Log Service, SAI-AIS-LOG-A.02.01
http://www.saforum.org/specification/AIS_information
- [2] Java Logging Technology
<http://java.sun.com/javase/6/docs/technotes/guides/logging/index.html>

1.4 Abbreviations

AIS	Application Interface Specification
API	Application Programming Interface
EE	Enterprise Edition
J2SE	Java Platform 2, Standard Edition
JVM	Java Virtual Machine
SE	Standard Edition

1.5 Java Logging Overview

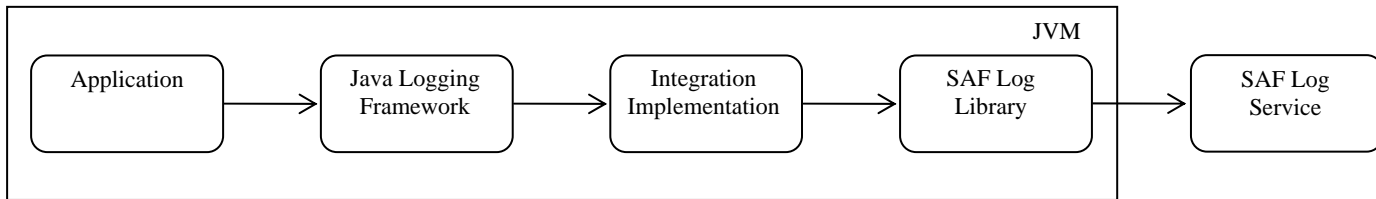
The following image illustrates the Java Logging API support for logging in Java.



- **Logger**
Applications make logging calls on *Logger* objects. These *Logger* objects allocate *LogRecord* objects which are passed to *Handler* objects for publication. Both *Loggers* and *Handlers* may use *Filters* to decide if they are interested in a particular *LogRecord*
- **Handler**
Handlers may direct output to other *Handlers* and it can use a *Formatter* to localize and format the message before publishing.

2 Architecture

The image below illustrates how a Java application emitting Java log records will be supported by the SAF Log Service. The Java application is only using the Java logging framework. An Integration Implementation is plugged into the Java logging framework as a backend that will transform the Java log records to SAF Log records and then forward them to the SAF Log Service.



3 Integration Requirements

3.1 Environment Requirements

The Integration Implementation shall be executable on J2SE 1.5 or later.

3.2 Life Cycle Requirements

At JVM start-up, the Integration Implementation shall initialize the SAF Log library and open one or several SAF log streams for use by the Java applications. The integration implementation will open SAF application log streams (or the system log stream) with the names according to configuration settings. The Integration Implementation shall also create one or several instances of a subclass to *java.util.logging.Handler*. Each log handler instance shall handle the log records to one SAF log stream. The log handler instances shall be added to the *java.util.logging.Logger* objects with the names listed in the configuration for each SAF log stream.

At JVM shut-down, the Integration Implementation shall close the log streams and finalize the SAF Log library.

3.3 Configuration Requirements

The integration implementation shall be configured with one entry for each SAF application (or system) log stream. Every entry shall contain a log stream name with the file creation attributes and a list of Java logger names to be mapped to this stream. If the application would like to use the system log stream in SAF, the predefined name in SAF should be used. The root logger, which emits all log records (if default Java configuration), has the name of an empty string ("").

The Integration Implementation shall be able to handle configuration changes at runtime and adjust the behavior accordingly.

3.4 Feature Requirements

The Integration Implementation shall filter out unwanted log records according to the selected log level before they are forwarded to the SAF Log Service. Filtering must be done before formatting in order to ensure that discarded records do not consume processing resources by formatting them. As the SAF Log Service is a pure log service and Java logging is a trace and log combination only higher log levels on the Java side should be forwarded to the SAF Log Service.

The SAF Log Service selects the log levels to be logged and calls the Integration Implementation to request it to log only at these levels. The lowest of these allowed levels will be mapped to a Java log level. Log records with a lower level than this level will be filtered out by the Integration Implementation. The SAF log levels shall be mapped on the Java log levels according to the following table.

SaLogSeverityT	java.util.logging.Level
SA_LOG_LEVEL_SEV_EMERGENCY	OFF
SA_LOG_LEVEL_SEV_ALERT	OFF
SA_LOG_LEVEL_SEV_CRITICAL	SEVERE
SA_LOG_LEVEL_SEV_ERROR	SEVERE
SA_LOG_LEVEL_SEV_WARNING	WARNING
SA_LOG_LEVEL_SEV_NOTICE	WARNING
SA_LOG_LEVEL_SEV_INFO	INFO

It is still possible to have another *Handler* that will produce log records only for the Java parts and this handler can use a different level than the log records that are forwarded to the SAF Log Service.

The Integration Implementation shall format log records to be forwarded, if this is requested by the application. The message is localized, if the log record has been associated with a resource bundle, and the message is parameterized with the given parameters, if it contains any argument pattern. It is assumed that the raw message is in the *java.text.MessageFormat* style. The localizing and the parameterizing are typically performed by the built-in method *java.util.Formatter.formatMessage()*.

The Integration Implementation shall forward log records to the SAF Log Service by using a *SaLogRecordT* parameter with the following mappings to the Java *java.util.logging.LogRecord*.

LogRecord	Type	SaLogRecordT	Remark
millis	long	logTimeStamp	The Java field shall be multiplied with 1000.
-		logHdrType	SA_LOG_GENERIC_HEADER
-		notificationClassId	NULL
-		logSvcUserName	Not to be set. The default will be used.
level	Level SEVERE WARNING INFO CONFIG FINE FINER FINEST	logSeverity	SEVERE -> SA_LOG_SEV_CRITICAL WARNING -> SA_LOG_SEV_WARNING INFO -> SA_LOG_SEV_INFO Records of lower levels shall never be forwarded.
loggerName	String	NA	Application log stream name or <i>saLgStr=saLogSystem</i>
message, parameters	String, Object	logBuffer	The message shall be localized and parameterized with the parameters.

4 Application Requirements

4.1 Java SE/EE Specific

The following are requirements for Java SE applications, but not for Java EE applications.

Early when the application starts up, it shall initialize the Integration Implementation.

The application shall not use the logging feature before the Integration Implementation has completed the initialization.

Late when the application shuts down, it shall close the Integration Implementation.

If a localization resource bundle is not loadable by the system class loader, the context class loader of the current thread shall be set to the class loader able to load the bundle.

In a Java EE environment these tasks shall be performed by the container or by a container extension. The container shall not start the Java EE applications before the Integration Implementation has completed the initialization.

4.2 API

The application shall use the Java Logging API in the *java.util.logging* package as defined in J2SE 1.5.

4.3 Design Rules

The application shall use the log level *INFO*, *WARNING* or *SEVERE* for log entries intended to be forwarded to the SAF Log Service.

The application shall not format log messages but it shall use the parameterization features of the Java Logging API when needed.

If localization is wanted, the application shall use localization keys instead of message texts. The application must also associate the used logger with the resource bundle containing the mappings from localization keys to message strings.

It should be noted that in many cases it is inappropriate to localize log messages in this way, because it may consume considerable processing resources and because the human log readers may be from different countries. In these cases post-localization of the log messages offline should be preferred.

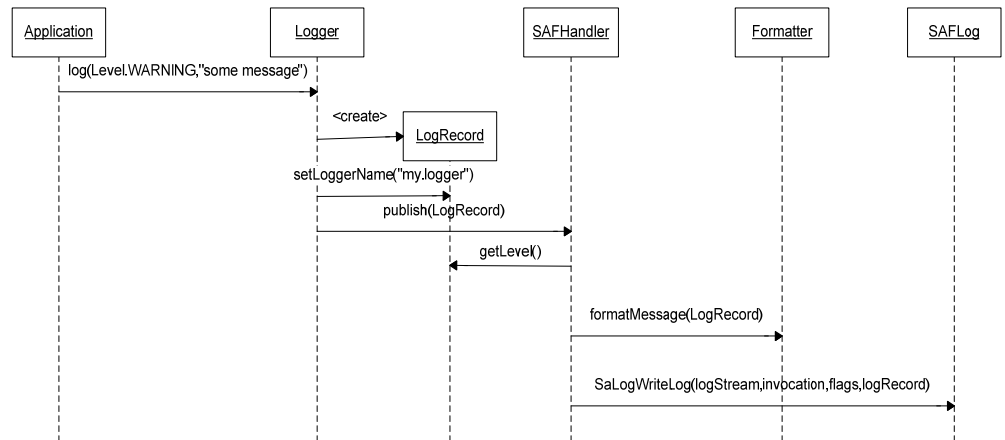
4.4 Artifacts

If localization is wanted, the application must provide a localization resource bundle.

5 Use Cases

5.1 The Application Generates a Wanted Log Record

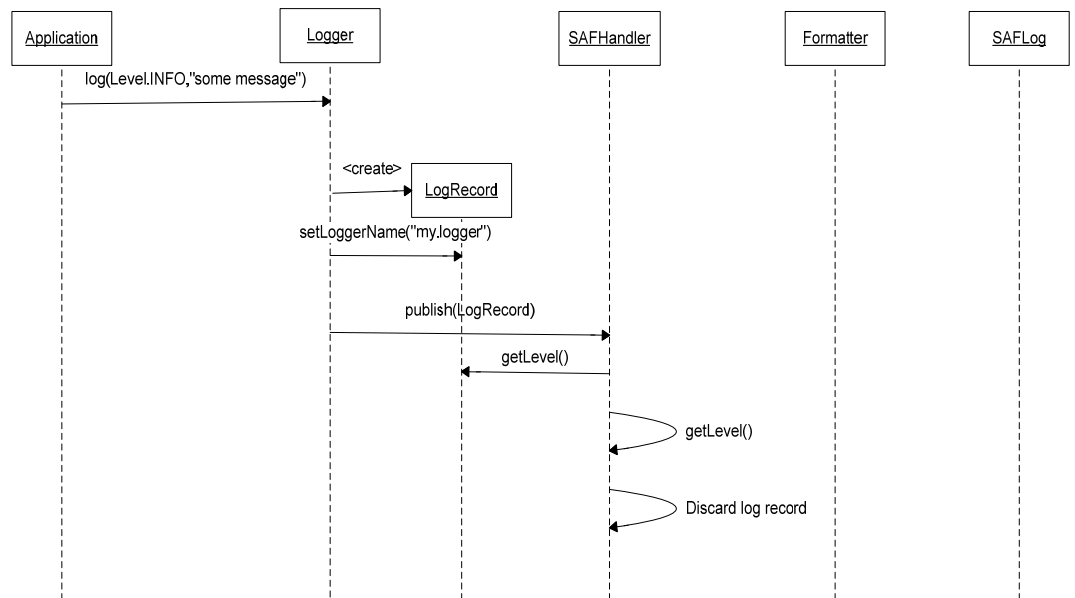
The following sequence diagram describes an application that generates a log record on a level that should be logged.



- The application uses the logger to write a log message
- The Logger creates the LogRecord and set the logger name for it.
- The Logger finds the appropriate handler for the log message and publish it.
- The integration implementation, SAFHandler, checks the level to make sure that this is a message that should be logged before transforming the log record.
- The formatter is used to create a localized string.
- The log record is forwarded to the SAF Log Service with the following parameters for the log record:
 - logTimeStamp from the Java LogRecord
 - logHdrTyp = SA_LOG_GENERIC_HEADER
 - SaLogGenericLogHeader
 - notificationClassId = NULL
 - logSvcUserName = default (will be the container name)
 - logSeverity = SA_LOG_SEV_WARNING
 - logBuffer = formatted log record from Java

5.2 The Application Generates an Unwanted Log Record

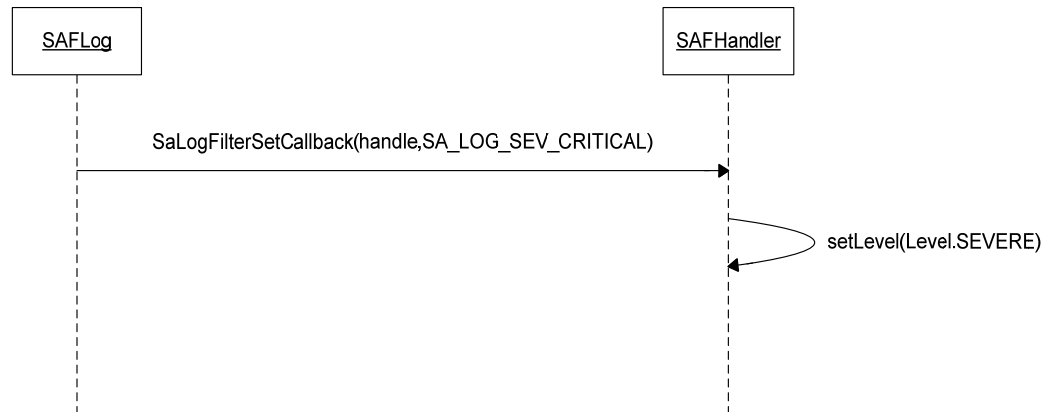
This scenario shows a sequence diagram when an application tries to log on a level that is not transferred to the SAF Log Service. The level at or above which log records are transferred to the SAF Log Service is WARNING.



- The application uses the normal Java logging API to log on level INFO
- The Logger creates the LogRecord and finds the appropriate Handler for it.
- The integration implementation, SAFHandler, checks the level of the log record and finds that it is lower than the levels that should be logged.
- The Handler discards the log record before transforming it.

5.3 The Administrator Changes the Severity Level of the Log

The following sequence diagram describes the use case when the administrator changes the severity level for the SAF Log Service to SA_LOG_SEV_CRITICAL.



- The SAF Log Service use the `SaLogFilterSetCallback` to change the severity level.
- The integration implementation, `SAFHandler`, changes the severity level to `Level.SEVERE`, which will cause SEVERE log records only to be forwarded.