

Java Usage of the SA Forum Notification 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 AIS Notification Service.

1.2 Concepts

- Integration Implementation
The implementation that integrates between SAF and Java. This part uses the low level Java 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

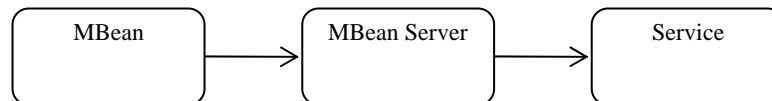
- [1] Service Availability Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.02.01
http://www.saforum.org/specification/AIS_Information/
- [2] Java Management Extensions (JMX) Specification, version 1.4
<http://jcp.org/en/jsr/detail?id=3>
- [3] Service Availability Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.03.01
http://www.saforum.org/specification/AIS_Information/

1.4 Abbreviations

AIS	Application Interface Specification
AMF	Availability Management Framework
API	Application Programming Interface
EE	Enterprise Edition
J2SE	Java Platform 2, Standard Edition
JMX	Java Management Extensions
JVM	Java Virtual Machine
NTF	Notification Service
SNMP	Simple Network Management Protocol
SE	Standard Edition

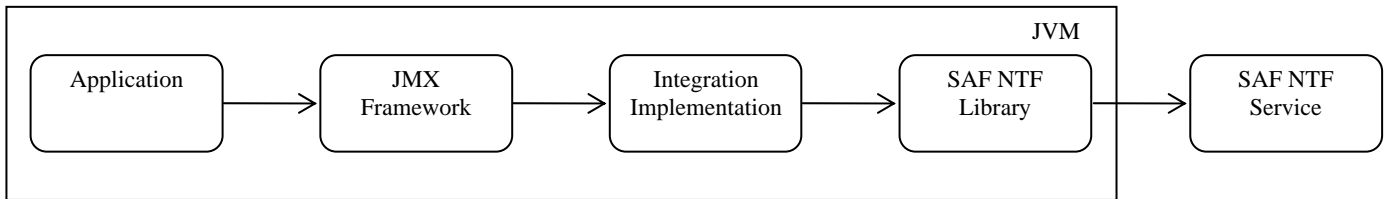
1.5 JMX Overview

The following image illustrates the JMX (Java Management Extensions) support for notifications in Java. MBeans represents resources and they are registered into an MBean server. MBeans may emit notifications that can be of user defined types or of a number of predefined types. Different services can find the emitting MBeans from the MBean server and they can subscribe to receive notifications emitted by these MBeans. The MBeans are part of the application while the MBean server is part of the JMX framework supported by all Java runtime environments.



2 Architecture

The image below illustrates how a Java application emitting JMX notifications will be supported by the Notification Service. The Java application is only using the JMX framework and design patterns. An Integration Implementation is plugged into the JMX framework as a backend that will transform the JMX notifications to NTF notifications and then forward them to the SAF NTF 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 NTF library. It shall also create or reuse an existing MBean server and subscribe for MBean registrations in order to later subscribe for JMX alarm notifications from MBeans when they are registered.

At JVM shut-down, the Integration Implementation shall finalize the SAF NTF library and unsubscribe for MBean registrations and alarm notifications.

3.3 Configuration Requirements

The Integration Implementation shall be configured with vendor names with the dotted string notification (string with dot separated components, which are the internet domain name of the vendor in reverse order). The vendor name shall be mapped to a unique 32 bit number, which is suggested to be the SNMP enterprise number. Such entries are needed for all vendor names used in alarm notifications (in the *subtype* field) emitted by the application, except if the vendor name consists of two components (e.g. "com.acme") and it may be mapped to a hashed number.

3.4 Feature Requirements

When MBeans capable to emit JMX alarm notifications are registered into the MBean server, the Integration Implementation shall subscribe for such notifications from these MBeans.

The Integration Implementation shall forward emitted JMX regular alarm and security alarm notifications to the SAF NTF Service.

The JMX regular alarm and security alarm notification fields (see section 4.3) shall be mapped to the NTF alarm and security alarm notification parameters according to the following table.

Notification	Type	SaNtfAlarm NotificationT	SaNtfSecurity Alarm NotificationT	Remark
type	String	eventType	eventType	<p>“org.saforum.ntf.alarm.communication” -> SA_NTF_ALARM_COMMUNICATION</p> <p>“org.saforum.ntf.alarm.qos” -> SA_NTF_ALARM_QOS</p> <p>“org.saforum.ntf.alarm.processing” -> SA_NTF_ALARM_PROCESSING</p> <p>“org.saforum.ntf.alarm.equipment” -> SA_NTF_ALARM_EQUIPMENT</p> <p>“org.saforum.ntf.alarm.environment” -> SA_NTF_ALARM_ENVIRONMENT</p> <p>“org.saforum.ntf.securityalarm.integrityviolation” -> SA_NTF_INTEGRITY_VIOLATION</p> <p>“org.saforum.ntf.securityalarm.operationalviolation” -> SA_NTF_OPERATION_VIOLATION</p> <p>“org.saforum.ntf.securityalarm.physicalviolation” -> SA_NTF_PHYSICAL_VIOLATION</p> <p>“org.saforum.ntf.securityalarm.securityserviceviolation” -> SA_NTF_SECURITY_SERVICE_VIOLATION</p> <p>“org.saforum.ntf.securityalarm.timeviolation” -> SA_NTF_TIME_VIOLATION</p>
object	ObjectName	notificationObject	notificationObject	The value of the ‘dn’ key is forwarded. If <i>object</i> is null, <i>source</i> is used instead.
source	Object	notifyingObject	notifyingObject	The value of the ‘dn’ key is forwarded.
subtype	String	notificationClassId	notificationClassId	The JMX <i>subtype</i> field is mapped to the NTF <i>notificationClassId</i> parameter according to rules explained below.
timeStamp	long	eventTime	eventTime	The <i>timeStamp</i> field shall be multiplied with 1000. If <i>timeStamp</i> is zero, <i>eventTime</i> is set to SA_TIME_UNKNOWN.
-		notificationId	notificationId	The NTF parameter value is allocated when the notification is sent to NTF. When an alarm is raised and the JMX <i>alarmId</i> field is present, the Integration Implementation must save the mapping between the JMX <i>alarmId</i> field and the NTF <i>notificationId</i> parameter.
-		correlated Notifications	correlated Notifications	When an alarm is modified or cleared and the JMX <i>alarmId</i> field is present, the Integration Implementation must insert the <i>notificationId</i> of the notification that raised the alarm. The value is decided from the <i>alarmId</i> value of the current notification and the saved mapping between <i>alarmId</i> and <i>notificationId</i> .
message	String	additionalText	additionalText	The message shall be parameterized with the parameters from <i>additionalInfo</i> . The message is

				in the <i>java.text.MessageFormat</i> style. If <i>resourceBundleName</i> is present, the message shall be localized first.
-		probableCause	probableCause	The NTF parameter is set to SA_NTF_UNSPECIFIED_REASON.
severity	Integer	perceivedSeverity	severity	CLEARED -> SA_NTF_SEVERITY_CLEARED INDETERMINATE -> SA_NTF_SEVERITY_INDETERMINATE WARNING -> SA_NTF_SEVERITY_WARNING MINOR -> SA_NTF_SEVERITY_MINOR MAJOR -> SA_NTF_SEVERITY_MAJOR CRITICAL -> SA_NTF_SEVERITY_CRITICAL
additionalInfo	Object[]	additionalInfo	additionalInfo	The array elements are mapped to <i>SaNtfAdditionalInfoT</i> structures with <i>infoId</i> set to the index value and <i>infoType</i> mapped as follows. Boolean -> SA_NTF_VALUE_UINT8 Byte -> SA_NTF_VALUE_INT8 Short -> SA_NTF_VALUE_INT16 Integer -> SA_NTF_VALUE_INT32 Long -> SA_NTF_VALUE_INT64 Float -> SA_NTF_VALUE_FLOAT Double -> SA_NTF_VALUE_DOUBLE String -> SA_NTF_VALUE_STRING The value of <i>infoValue</i> is the same as the primitive value of the Java object except for Boolean, where <i>false</i> and <i>true</i> are mapped on 0 and 1 respectively, and String, where '\0' must be appended to the char array.
detector	String	-	securityAlarm Detector	
serviceUser	String	-	serviceUser	
serviceProvider	String	-	serviceProvider	

In order to map the *subtype* string to the three integers (*vendorId*, *majorId* and *minorId*) of *notificationClassId* the procedure is as follows.

The *subtype* string consists of a number of components separated by dots, e.g. "uk.co.acme.myapp.connection.setupfailure". The first part of the string (two or more components) shall determine *vendorId*; the middle part (one component) shall determine *majorId* and the last part (one or more components) shall determine *minorId*. The values of *majorId* and *minorId* are decided as the least significant 16 bits of the hash codes (as returned by the *hashCode* method) of the corresponding string parts. The value zero is used if any string part is lacking, i.e. if the string ends prematurely.

First the vendor part of the *subtype* string is identified by matching the beginning of the string with the configured entries of vendors. Each entry in the configuration contains a vendor name (in the dotted string notification). The vendor names of the configuration entries are compared to the beginning of the *subtype* string, one after another until a match is found. If a match is found, the mapped number of the entry is used for *vendorId*. If no match is found, the vendor name is assumed to consist of the first two components (until the second separating dot) and the hash code of this name is used for *vendorId*. E.g. for a *subtype* of “*uk.co.acme.myapp.connection.setupfailure*” and with a configuration entry containing “*uk.co.acme*” mapped to 314, a match will be found and 314 will be used for *vendorId*.

Then the part of the *subtype* string deciding *majorId* is identified as the string component following the vendor name and its separating dot. It ends at the next separating dot. The previously mentioned example will result in that the least significant 16 bits of the hash code of “*myapp*” will be used for *majorId*.

At last the part of the *subtype* string deciding *minorId* is identified as the string remaining after the part deciding *majorId* and its separating dot. The example will result in that the least significant 16 bits of the hash code of “*connection.setupfailure*” will be used for *minorId*.

If *resourceBundleName* is present in the notification, *message* shall be localized. The localization resource bundle shall be loaded by the context class loader of the current thread. If that does not succeed, the system class loader shall be used instead.

If there is no ‘*dn*’ key in the JMX object name when deciding the *notificationObject*, the value returned by the AMF *saAmfComponentNameGet()* function shall be used instead as the *notificationObject* value.

When alarms are raised, the Integration Implementation must save the mapping between the JMX *alarmId* value and the NTF *notificationId* value, if the *alarmId* field is present. The mapping can be discarded when the alarm is cleared. The saved mappings of ids are used for deciding the values of *correlatedNotifications* when alarms are modified or cleared.

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 access the MBean server before the Integration Implementation has completed the initialization, at which time the MBean server is ready.

The application shall get the MBean server used for alarm notifications from the Integration Implementation.

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. The container shall provide a reference to the MBean server used for alarm notifications as an entry in the JNDI naming environment of the application components. The entry shall have the name *jmx/_server* in the *java:comp/env* naming context.

4.2 API

The application shall use the Java Management Extensions in the *javax.management* package as defined in J2SE 1.5.

4.3 Design Rules

The application shall emit JMX notifications when raising, modifying or clearing a regular alarm or a security alarm. The notifications shall be instances of the *javax.management.Notification* class with the following contents.

Field	Type	Meaning	Remark
source	Object	The object name of the emitting MBean or a reference to the MBean.	Mandatory. The object name shall contain the key 'dn' associated with the value of the distinguished name required by NTF.
type	String	The event type as a string expressed in dot notation. The value shall be one of the following for regular alarms: "org.saforum.ntf.alarm.communication" "org.saforum.ntf.alarm.qos" "org.saforum.ntf.alarm.processing" "org.saforum.ntf.alarm.equipment" "org.saforum.ntf.alarm.environment" The value shall be one of the following for security alarms: "org.saforum.ntf.securityalarm.integrityviolation" "org.saforum.ntf.securityalarm.operationalviolation" "org.saforum.ntf.securityalarm.physicalviolation" "org.saforum.ntf.securityalarm.securityserviceviolation" "org.saforum.ntf.securityalarm.timeviolation"	Mandatory.
sequenceNumber	long	The notification sequence number within the source object. It is a serial number identifying a particular instance of the notification in the context of the notification source.	Optional.
timeStamp	long	The notification timestamp indicates when the notification was generated.	Optional.
message	String	The message could be the explanation of the notification for display to a user. It may include parameters in the <i>java.text.MessageFormat</i> style, where the parameters are contained in <i>additionalInfo</i> . If localization is wanted, the message shall be a localization key.	Optional.
userData	Object	This is used for whatever additional information the notification source wants to communicate to its consumers.	Mandatory. The contents of this field is defined below.

The *userData* field shall be a *javax.management.openmbean.CompositeData* implementation with the following contents.

Key	Type	Meaning	Remark
object	ObjectName	The object which this alarm is valid for.	Optional. If absent, the value of <i>source</i> is implied. The object name shall contain the key ' <i>dn</i> ' associated with the value of the distinguished name required by NTF.
subtype	String	The notification subtype groups the notifications into vendor defined subtypes, where each subtype contains notifications for similar situations. It is used for filtering and for selecting a suitable text description. It is a string expressed in dot notation.	Mandatory. It must be a unique subtype identifier by using the dot notation beginning with the vendor domain name (in reverse order) and followed by additional components.
severity	Integer	CLEARED = 0 INDETERMINATE = 1 WARNING = 2 MINOR = 3 MAJOR = 4 CRITICAL = 5	Mandatory.
alarmId	String	This is an identifier for a particular alarm instance. The identifier must be unique during the lifetime of the alarm instance.	Optional.
additionalInfo	Object[]	Additional information that can be used to parameterize <i>message</i> and to convey specific information that does not fit into the other fields.	Optional. Allowed object types in the array are Boolean, Byte, Short, Integer, Long, Float, Double and String.
detector	String	The detector of the security alarm.	Not used for regular alarms. Mandatory for security alarms.
serviceUser	String	The service user whose request for service led to the generation of this security alarm.	Not used for regular alarms. Mandatory for security alarms.
serviceProvider	String	The intended service provider of the service that led to the generation of this security alarm.	Not used for regular alarms. Mandatory for security alarms.
resourceBundleName	String	The resource bundle to use for localizing messages.	Optional. Only used if localization is wanted.

When raising or modifying regular alarms, the application shall provide at least *source*, *type*, *subtype* and *severity* with the value INDETERMINATE, WARNING, MINOR, MAJOR or CRITICAL. When raising or modifying security alarms, the application shall provide the same contents and additionally *detector*, *serviceUser* and *serviceProvider*.

When clearing regular alarms, the application shall provide at least *source*, *type*, *subtype* and *severity* with the value CLEARED. When clearing security alarms, the application shall provide the same contents and additionally *detector*, *serviceUser* and *serviceProvider*.

When modifying or clearing an alarm, the alarm instance is identified by the values of *object* (or *source*, if *object* is not present), *type*, *subtype* and *alarmId*, if it is present.

The object names shall contain the key '*dn*' associated with the value of the distinguished name required by NTF.

If localization is wanted, the application shall use localization keys instead of message texts. The application must also include in the notification the name of the localization resource bundle containing the mappings from localization keys to message strings.

4.4 Artifacts

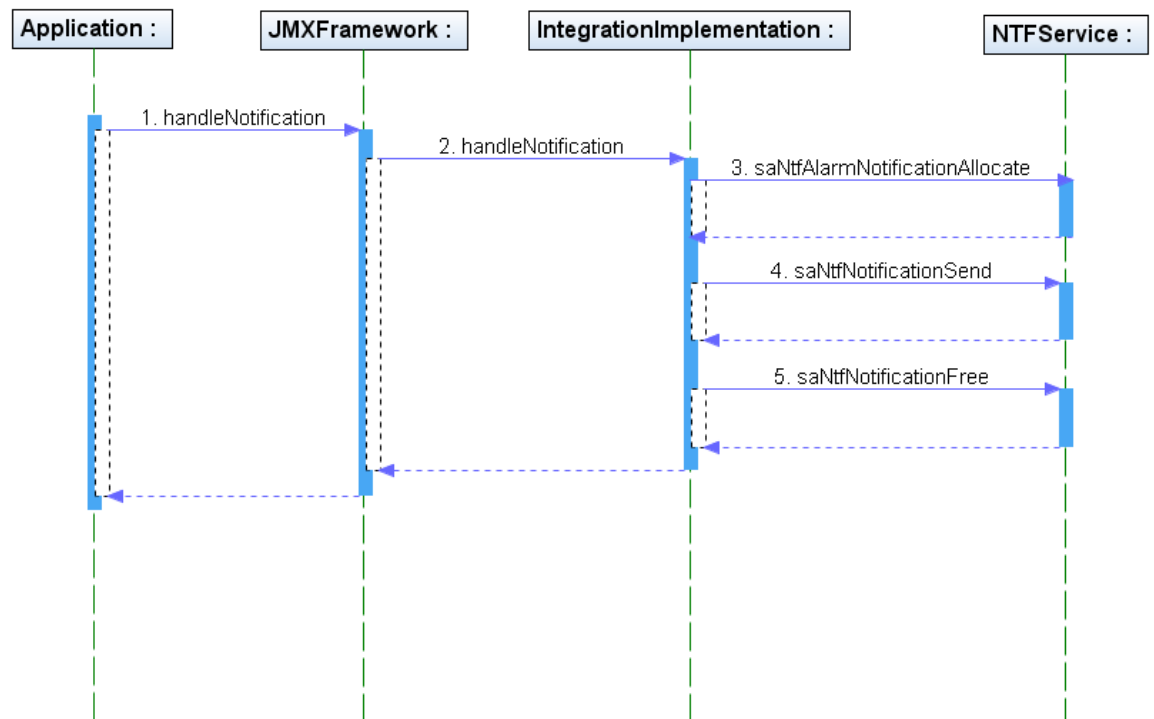
The application must provide a list of used alarms with their data (at least type and subtype).

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

5 Use Cases

5.1 The Application Raises an Alarm

The following sequence diagram describes how an application raises an alarm.

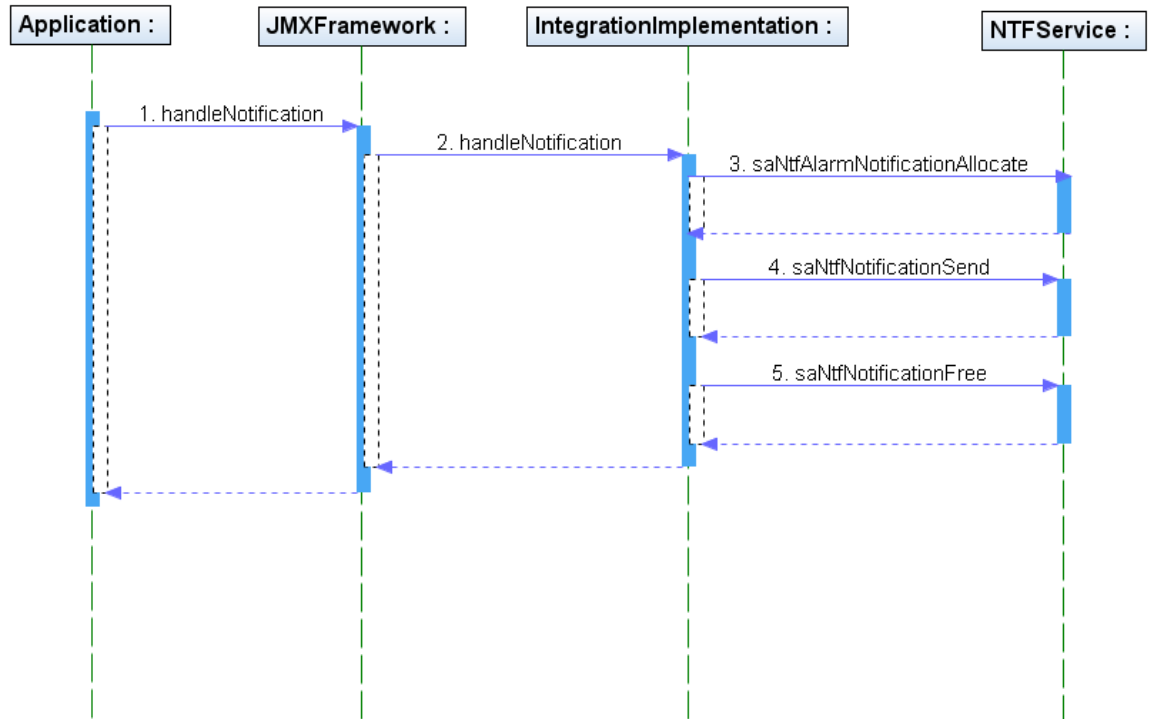


1. The application MBean sends a JMX notification to its subscribers. It is an alarm notification with *severity* MINOR and with *alarmId* set to a unique value.
2. The JMX Framework replaces the *source* with the object name of the emitting MBean before forwarding the notification to the Integration Implementation.
3. The Integration Implementation allocates a memory area for the NTF notification and builds the notification by mapping the values from the JMX notification.
4. The Integration Implementation sends the NTF notification to the SAF NTF Service and a value for *notificationId* is allocated. The mapping from *alarmId* to *notificationId* is saved by the Integration Implementation in a table.

5. The Integration Implementation frees the memory area used for the NTF notification.

5.2 The Application Clears an Alarm

The following sequence diagram describes how an application clears an alarm.



1. The application MBean sends a JMX notification to its subscribers. It is an alarm notification with *severity* CLEARED and with *alarmId* set to the same value as in the notification raising the alarm.
2. The JMX Framework replaces the *source* with the object name of the emitting MBean before forwarding the notification to the Integration Implementation.
3. The Integration Implementation allocates a memory area for the NTF notification and builds the notification by mapping the values from the JMX notification. The value of *correlatedNotifications* is decided by using *alarmId* of the current notification to look up the *notificationId* of the notification that raised the alarm in the saved mappings table.

4. The Integration Implementation sends the NTF notification to the SAF NTF Service. The previously saved mapping from *alarmId* to *notificationId* is discarded by the Integration Implementation.
5. The Integration Implementation frees the memory area used for the NTF notification.