Service Availability™ Forum Application Interface Specification

Volume 5: Event Service SAI-AIS-EVT-B.01.01

The Service Availability™ solution is high availability and more; it is the delivery of ultra-dependable communication services on demand and without interruption.

This Service Availability™ Forum Application Interface Specification document might contain design defects or errors known as errata, which might cause the product to deviate from published specifications. Current characterized errata are available on request.

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

The Service Availability™ Specification(s) (the "Specification") found at the URL http://www.saforum.org (the "Site") is generally made available by the Service Availability Forum (the "Licensor") 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 Specification are set forth in this agreement (this "Agreement").

IMPORTANT – PLEASE READ THE TERMS AND CONDITIONS PROVIDED IN THIS AGREEMENT BEFORE DOWNLOADING OR COPYING THE SPECIFICATION. IF YOU AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT, CLICK ON THE "ACCEPT" BUTTON. BY DOING SO, YOU AGREE TO BE BOUND BY THE TERMS AND CONDITIONS STATED IN THIS AGREEMENT. IF YOU DO NOT WISH TO AGREE TO THESE TERMS AND CONDITIONS, YOU SHOULD PRESS THE "CANCEL"BUTTON AND THE DOWNLOAD PROCESS WILL NOT PROCEED.

1. LICENSE GRANT. Subject to the terms and conditions of this Agreement, Licensor hereby grants you a non-exclusive, worldwide, non-transferable, revocable, but only for breach of a material term of the license granted in this section 1, fully paid-up, and royalty free license to:

a. reproduce copies of the Specification to the extent necessary to study and understand the Specification and to use the Specification to create products that are intended to be compatible with the Specification;

b. distribute copies of the Specification to your fellow employees who are working on a project or product development for which this Specification is useful; and

c. distribute portions of the Specification as part of your own documentation for a product you have built, which is intended to comply with the Specification.

2. DISTRIBUTION. If you are distributing any portion of the Specification in accordance with Section 1(c), your documentation must clearly and conspicuously include the following statements:

a. Title to and ownership of the Specification (and any portion thereof) remain with Service Availability Forum ("SA Forum").

b. The Specification is provided "As Is." SAF makes no warranties, including any implied warranties, regarding the Specification (and any portion thereof) by Licensor.

c. SAF shall not be liable for any direct, consequential, special, or indirect damages (including, without limitation, lost profits) arising from or relating to the Specification (or any portion thereof).

d. The terms and conditions for use of the Specification are provided on the SA Forum website.

3. RESTRICTION. Except as expressly permitted under Section 1, you may not (a) modify, adapt, alter, translate, or create derivative works of the Specification, (b) combine the Specification (or any portion thereof) with another document, (c) sublicense, lease, rent, loan, distribute, or otherwise transfer the Specification to any third party, or (d) copy the Specification for any purpose.

4. NO OTHER LICENSE. Except as expressly set forth in this Agreement, no license or right is granted to you, by implication, estoppel, or otherwise, under any patents, copyrights, trade secrets, or other intellectual property by virtue of your entering into this Agreement, downloading the Specification, using the Specification, or building products complying with the Specification.

5. OWNERSHIP OF SPECIFICATION AND COPYRIGHTS. The Specification and all worldwide copyrights therein are the exclusive property of Licensor. You may not remove, obscure, or alter any copyright or other proprietary rights notices that are in or on the copy of the Specification you download. You must reproduce all such notices on all copies of the Specification you make. Licensor may make changes to the Specification, or to items referenced therein, at any time without notice. Licensor is not obligated to support or update the Specification.

6. WARRANTY DISCLAIMER. THE SPECIFICATION IS PROVIDED "AS IS." LICENSOR DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF THIRD-PARTY RIGHTS, FITNESS FOR ANY PARTICULAR PURPOSE, OR TITLE. Without limiting the generality of the foregoing, nothing in this Agreement will be construed as giving rise to a warranty or representation by Licensor that implementation of the Specification will not infringe the intellectual property rights of others.

7. PATENTS. Members of the Service Availability Forum and other third parties [may] have patents relating to the Specification or a particular implementation of the Specification. You may need to obtain a license to some or all of these patents in order to implement the Specification. You are responsible for determining whether any such license is necessary for your implementation of the Specification and for obtaining such license, if necessary. [Licensor does not have the authority to grant any such license.] No such license is granted under this Agreement.

8. LIMITATION OF LIABILITY. To the maximum extent allowed under applicable law, **LICENSOR DISCLAIMS ALL LIABILITY AND DAMAGES, INCLUDING DIRECT, INDIRECT, CONSEQUENTIAL, SPECIAL, AND INCIDENTAL DAMAGES, ARISING FROM OR RELATING TO THIS AGREEMENT, THE USE OF THE SPECIFICATION OR ANY PRODUCT MANUFACTURED IN ACCORDANCE WITH THE SPECIFICATION, WHETHER BASED ON CONTRACT, ESTOPPEL, TORT, NEGLIGENCE, STRICT LIABILITY, OR OTHER THEORY. NOTWITHSTANDING ANYTHING TO THE CONTRARY, LICENSOR'S TOTAL LIABILITY TO YOU ARISING FROM OR RELATING TO THIS AGREEMENT OR THE USE OF THE SPECIFICATION OR ANY PRODUCT MANUFACTURED IN ACCORDANCE WITH THE SPECIFICATION WILL NOT EXCEED ONE HUNDRED DOLLARS ($100). YOU UNDERSTAND AND AGREE THAT LICENSOR IS PROVIDING THE SPECIFICATION TO YOU AT NO CHARGE AND, ACCORDINGLY, THIS LIMITATION OF LICENSOR'S LIABILITY IS FAIR, REASONABLE, AND AN ESSENTIAL TERM OF THIS AGREEMENT.**

9. TERMINATION OF THIS AGREEMENT. Licensor may terminate this Agreement, effective immediately upon written notice to you, if you commit a material breach of this Agreement and do not cure the breach within ten (30) days after receiving written notice thereof from Licensor. Upon termination, you will immediately cease all use of the Specification and, at Licensor's option, destroy or return to Licensor all copies of the Specification and certify in writing that all copies of the Specification have been returned or destroyed. Parts of the Specification that are included in your product documentation pursuant to Section 1 prior to the termination date will be exempt from this return or destruction requirement.

10. ASSIGNMENT. You may not assign, delegate, or otherwise transfer any right or obligation under this Agreement to any third party without the prior written consent of Licensor. Any purported assignment, delegation, or transfer without such consent will be null and void.

11. GENERAL. This Agreement will be construed in accordance with, and governed in all respects by, the laws of the State of Delaware (without giving effect to principles of conflicts of law that would require the application of the laws of any other state). You acknowledge that the Specification comprises proprietary information of Licensor and that any actual or threatened breach of Section 1 or 3 will constitute immediate, irreparable harm to Licensor for which monetary damages would be an inadequate remedy, and that injunctive relief is an appropriate remedy for such breach. All waivers must be in writing and signed by an authorized representative of the party to be charged. Any waiver or failure to enforce any provision of this Agreement on one occasion will not be deemed a waiver of any other provision or of such provision on any other occasion. This Agreement may be amended only by binding written instrument signed by both parties. This Agreement sets forth the entire understanding of the parties relating to the subject matter hereof and thereof and supersede all prior and contemporaneous agreements, communications, and understandings between the parties relating to such subject matter

Table of Contents — Volume 5, Event Service

1 Document Introduction

1.1 Document Purpose

This document defines the Event Service of the Application Interface Specification (AIS) of the Service Availability™ 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) and with the Service Availability™ Forum System Management Specification, which is still under development.

1.2 AIS Documents Organization

The Application Interface Specification is organized into the following volumes:

Volume 1, the Overview document, provides a brief guide to the remainder of the Application Interface Specification. It describes the objectives of the AIS specification as well as programming models and definitions that are common to all specifications. Additionally, it contains an overview of the Availability Management Framework and of the other services as well as the system description and conceptual models, including the physical and logical entities that make up the system. Volume 1 also contains a chapter that describes the main abbreviations, concepts and terms used in the AIS documents.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisOverview.B0101.pdf

Volume 2 describes the Availability Management Framework API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisAmf.B0101.pdf

Volume 3 describes the Cluster Membership Service API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisClm.B0101.pdf

Volume 4 describes the Checkpoint Service API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisCkpt.B0101.pdf

Volume 5 (this volume) describes the Event Service API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisEvt.B0101.pdf

Volume 6 describes the Message Service API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisMsg.B0101.pdf

Volume 7 describes the Lock Service API.
The name of the pdf file containing this document for the AIS version B.01.01 is:
aisLck.B0101.pdf

1.3 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.

1.4 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.

1.5 Additional Information

1.5.1 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**).

1.5.2 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 Event Service within the Application Interface Specification (AIS).

2.1 Event Service

The Event Service is a publish/subscribe multipoint-to-multipoint communication mechanism that is based on the concept of event channels, where a publisher communicates asynchronously via events with one or more subscribers over an event channel.

Events consist of a standard header and zero or more bytes of publisher event data.

Multiple publishers and multiple subscribers can communicate over the same event channel. Individual publishers and individual subscribers can communicate over multiple event channels. Subscribers are anonymous, which means that they may join and leave an event channel at any time without involving the publisher(s).

3 SA Event Service API

3.1 Event Service Model

3.1.1 Events

An **event** consists of a standard set of **event attributes** (also referred to as the **event header**) and zero or more bytes of **event data**.

An event header is allocated using the *saEvtEventAllocate()* function and is released using the *saEvtEventFree()* function. The *saEvtEventAllocate()* function returns a handle that can be used in subsequent invocations of the functions of the Event Service API.

The event attributes are written and read using set and get functions of the Event Service API, rather than directly.

An event is published by invoking the *saEvtEventPublish()* function and specifying as parameters the event handle and optional additional information, the event data, contained in a separate free-form data buffer. Thus, a published event consists of the event header, containing the set of attributes, and optional additional information contained in the data buffer.

3.1.2 Event Channels

An event channel enables multiple publishers to communicate with multiple subscribers. It is global to a cluster and is identified by a unique name. To use the Event Service, a process must open an event channel via the *saEvtEventChannelOpen()* function or the *saEvtEventChannelOpenAsync()* function. The process can specify in the open call whether it wants only to access an existing event channel or whether the event channel is first to be created if it does not yet exist.

A process can open an event channel to publish events and to subscribe to receive events. Publishers can also be subscribers on the same event channel. Event channels can be deleted via the *saEvtChannelUnlink()* function.

Once an event has been allocated for an event channel via the *saEvtEventAllocate()* call, it can be published several times on the same event channel, possibly by changing its attributes prior to each publication.

An event channel is required to satisfy the following properties:

- **Best effort delivery** - The Event Service provides best effort delivery of events to an anonymous set of subscribers. A published event might be lost or might be delivered to a proper subset of the subscribers, that is, some subscribers might get the event while others do not. For example, there is no guarantee that an event is delivered to all existing subscribers, if the publisher fails while publishing the event. Moreover, a subscriber might lose events, if the subscriber node is overwhelmed with events or if the subscriber is slow to process events.
- **At most once delivery** - The Event Service must not deliver the same event for a particular subscription of a particular subscriber multiple times.
- **Event priority** - Events are published with a certain priority. High priority events are delivered to subscribers ahead of low priority events. In case of overflow, low priority events are discarded from the subscriber queues to make room for high priority events.
- **Event ordering** - At a given priority level, events sent by a given publisher are received by subscribers in the order in which the publisher published the events.
- **Event completeness** - Only complete events are published and delivered to subscribers.
- **Retention time** - Events published with a non-zero retention time are kept for the specified duration. This gives the opportunity for new subscribers to obtain events that had been published before their subscription on the event channel. Processes may use the functions of the Event Service API to remove events explicitly before the retention time expires.

An event channel may optionally support the following property:

- **Persistence** - Published events may be persisted to disk and may survive node failure or shutting down the entire cluster, but that is not required by this specification.

The Event Service API does not impose a specific layout for the published event data. Publishers and subscribers on an event channel must agree on the structure of the data for events published on that event channel and may use data marshalling if heterogeneity support is desired. Conventions on the structure of the event data may vary from one event channel to another.

To support heterogeneity of data representation between publishers and subscribers, an implementation of the Event Service should use data marshalling for event attributes contained in the event header.

A process subscribes to receive events on an event channel by invoking the *saEvtEventSubscribe()* function of the Event Service API. The Event Service delivers events to a subscribing process using the *saEvtEventDeliverCallback()* function of that process. To stop receiving events for which it has registered, a subscriber can invoke the *saEvtEventUnsubscribe()* function to unsubscribe for those events.

If a process terminates abnormally, the Event Service automatically closes all of its open event channels.

Some API functions return an error if limits imposed by the configuration of the Event Service are exceeded. For instance, the *saEvtEventPublish()* function returns: "SA_AIS_ERR_TOO_BIG - The *eventDataSize* or the total size of the event is larger than the maximum permitted value.".

The interfaces of the Event Service configuration are outside the scope of this specification.

3.1.3 Event Filtering

The standard set of event attributes includes an array of event patterns. The values of these patterns are set by the event publisher and are typically used to organize events into various categories. All users (publishers and subscribers) of an event channel must share the same conventions regarding the number of patterns being used, their ordering and contents, as well as meaning.

For example, an event channel used to notify changes made to a relational database could define events where only three patterns are being used, as follows:

- The first pattern contains the name of the database being modified.
- The second pattern contains the name of the table being modified.
- The third pattern contains the primary key of the record being modified.

The event data can be used to provide a description of the modified fields and the old/new values.

Event patterns play an important role in the Event Service, as they are the basis for filtering which events must be delivered to a particular subscriber.

When subscribing on an event channel, a process must specify which filters to apply on published events. Only events which satisfy the provided filters are delivered to the process. Refer to Section 3.3.6.3 on page 21 for a description of the filtering process.

Using the previous example of the database notifications published on an event channel, a subscriber can provide a filter array indicating:

- The name of a database the subscriber is interested in.
- The name of a table the subscriber is interested in.
- No filter for the primary key.

In this case, the process will receive all notification events related to the specified table in the specified database for any primary key.

3.2 Include File and Library Name

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

#include <saEvt.h>

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

libSaEvt.so

3.3 Type Definitions

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

3.3.1 Handles

3.3.1.1 SaEvtHandleT

typedef SaUint64T SaEvtHandleT;

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

3.3.1.2 SaEvtEventHandleT

typedef SaUint64T SaEvtEventHandleT;

The type of a handle to an event.

3.3.1.3 SaEvtChannelHandleT

typedef SaUint64T SaEvtChannelHandleT;

The type of a handle to an open event channel.

3.3.2 SaEvtSubscriptionIdT

typedef SaUint32T SaEvtSubscriptionIdT;

The type of an identifier for a particular **subscription** by a particular process on a particular event channel. This identifier is used to associate delivery of events for that subscription to the process.

3.3.3 SaEvtCallbacksT

The *SaEvtCallbacksT* structure is defined as follows:

```
typedef struct {
    SaEvtChannelOpenCallbackT saEvtChannelOpenCallback;
    SaEvtEventDeliverCallbackT saEvtEventDeliverCallback;
} SaEvtCallbacksT;
```

The callbacks structure supplied by a process to the Event Service that contains the callback functions that the Event Service can invoke.

3.3.4 SaEvtChannelOpenFlagsT

```
#define SA_EVT_CHANNEL_PUBLISHER 0X1
#define SA_EVT_CHANNEL_SUBSCRIBER 0X2
#define SA_EVT_CHANNEL_CREATE 0X4
typedef SaUint8T SaEvtChannelOpenFlagsT;
```

The *SaEvtChannelOpenFlagsT* type has the following interpretation:

- SA_EVT_CHANNEL_PUBLISHER - Open the event channel for publishing events.
- SA_EVT_CHANNEL_SUBSCRIBER - Open the event channel for subscribing for events.
- SA_EVT_CHANNEL_CREATE - Create an event channel if one does not already exist.

When an event channel is opened using the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions, some combination of these flags are bitwise ORed together to provide the value of the *channelOpenFlags*.

3.3.5 Event Patterns and Attributes

The *SaEvtEventPatternT* structure is defined below. An **event pattern** may contain a name (e.g., process name, checkpoint name, service instance name, and so on).

Alternatively, an event pattern may characterize an event (e.g., timedOut, newComponent, overload, and so on).

3.3.5.1 SaEvtEventPatternT

typedef struct {

SaSizeT allocatedSize;

SaSizeT patternSize;

*SaUint8T *pattern;*

} SaEvtEventPatternT;

In the context of the *saEvtEventAttributesGet()* function, these fields are used as follow:

- *allocatedSize* (in): size of the buffer allocated to receive the pattern value
- *patternSize* (out): actual size of the pattern of the received event
- *pattern* (out): pointer to a buffer the pattern value will be copied to

In the context of the *saEvtEventAttributesSet()* or *saEvtEventSubscribe()* functions, these fields are used as follow:

- *allocatedSize*: ignored
- *patternSize* (in): actual size of the pattern
- *pattern* (in): pointer to a buffer the pattern value is taken from

3.3.5.2 SaEvtEventPatternArrayT

typedef struct {

SaSizeT allocatedNumber;

SaSizeT patternsNumber;

*SaEvtEventPatternT *patterns;*

} SaEvtEventPatternArrayT;

In the context of the *saEvtEventAttributesGet()* function, these fields are used as follow:

- *allocatedNumber* (in): number of entries allocated in the patterns buffer
- *patternsNumber* (out): actual number of patterns in the event
- *patterns* (out): pointer to a buffer the array of patterns will be copied to

In the context of the *saEvtEventAttributesSet()* function, these fields are used as follow:

- *allocatedNumber*: ignored
- *patternsNumber* (in): number of patterns in the patterns array
- *patterns* (in): pointer to the array of patterns

3.3.5.3 SaEvtEventPriorityT

#define SA_EVT_HIGHEST_PRIORITY 0

#define SA_EVT_LOWEST_PRIORITY 3

typedef SaUint8T SaEvtEventPriorityT;

3.3.5.4 SaEvtEventIdT

typedef SaUint64T SaEvtEventIdT;

The type of an event identifier. Values ranging from 0 to 1000 have special meanings and cannot be used by the Event Service to identify regular events.'

#define SA_EVT_EVENTID_NONE 0

Event identifier for an allocated but not yet published event.

#define SA_EVT_EVENTID_LOST 1

Event identifier for a “lost event”.

3.3.5.5 Event Attributes

A process has read access to all attributes of an event allocated through the *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* functions. A process has no access to the event attributes of a published event. The only exception is to discard a published event with non-zero retention time by clearing its retention time through the *saEvtEventRetentionTimeClear()* function. For each attribute in the following list, it is specified whether the process in which the event resides has write access or not. Some attributes are only writable by the Event Service. Additionally, the default values for an event allocated through the *saEvtEventAllocate()* function are given.

- **Event Pattern Array** - An array defined above by the *SaEvtEventPatternArrayT* structure.
  Write access is permitted.
  Default: no patterns
- **Event Priority** - An event priority is of the type *SaEvtEventPriorityT.* Event priorities range from SA_EVT_HIGHEST_PRIORITY to SA_EVT_LOWEST_PRIORITY.

Write access is permitted.
Default: SA_EVT_EVENT_LOWEST_PRIORITY

- **Event Publish Time** - The time when the event is published, which can be any time between the start and the end of the event publish API call. The Event Service fills in this time when the event is published.
  This is a read-only attribute.
  Default: SA_TIME_UNKNOWN
- **Event Retention Time** - The retention time is the duration for which the event is retained.
  Write access is permitted.
  Default: 0
- **Event Publisher Name** - The name of the entity that publishes an event on the event channel. Depending on the conventions associated with the event channel, this name may refer to the publishing process, the publishing component, and so on.
  Write access is permitted.
  Default: empty string (*SaNameT.length* = 0)
- **Event Id** - The cluster-wide unique identifier of the event on the event channel. It should not be assumed that event ids are consecutive or increasing. The event id attribute is set automatically by the Event Service when the event is published.
  This is a read-only attribute.
  Default: SA_EVT_EVENTID_NONE

3.3.6 Event Filters

The Event Service supports several different types of filters and pattern matching algorithms, as defined by the following enumeration type.

3.3.6.1 SaEvtEventFilterTypeT

```
typedef enum {
    SA_EVT_PREFIX_FILTER = 1,
    SA_EVT_SUFFIX_FILTER = 2,
    SA_EVT_EXACT_FILTER = 3,
    SA_EVT_PASS_ALL_FILTER = 4
} SaEvtEventFilterTypeT;
```

The values of the *saEvtEventFilterTypeT* enumeration type.
The corresponding pattern matching algorithms are explained later in Table 1 on page 22.

3.3.6.2 SaEvtEventFilterT

```
typedef struct {
    SaEvtEventFilterTypeT filterType;
    SaEvtEventPatternT filter;
} SaEvtEventFilterT;
```

The event filter structure defines the filter type and the filter pattern to be applied on an event pattern when filtering events on an event channel.

3.3.6.3 SaEvtEventFilterArrayT

```
typedef struct {
    SaSizeT filtersNumber;
    SaEvtEventFilterT *filters;
} SaEvtEventFilterArrayT;
```

The event filter array structure defines one or more filters.

Filters are passed to the Event Service by a subscriber process via the *saEvtEventSubscribe()* call. The Event Service does the filtering to decide whether a published event is delivered to a subscriber for a given subscription by matching the first filter (contents and type) against the first pattern in the event pattern array, the second filter against the second pattern in the event pattern array, and so on up to the last filter. An event matches a given subscription if all patterns of the event match all filters provided in an invocation of the *saEvtEventSubscribe()* call.

Table 1 Filter Types and Pattern Matching Algorithms

Filter Type	Matching Algorithm
SA_EVT_PREFIX_FILTER	The entire filter must match the first filterSize characters of the event pattern. Match example: Filter=”abcd”, Event Pattern=”abcdxyz” Match example: Filter=”abcd”, Event Pattern=”abcd” Match example: Filter=”XYz”, Event Pattern=”XYzaB” Non-Match example: Filter=”xyz”, Event Pattern=”abcdxyz” Non-Match example: Filter=”Xyz”, Event Pattern=”xyzab” Non-Match example: Filter=”xyz”, Event Pattern=”xy” (The entire filter does not match the first part of the pattern; only the first two characters match.)
SA_EVT_SUFFIX_FILTER	The entire filter must match the last filterSize characters of the event pattern. Match example: Filter=”xyz”, Event Pattern=”abcdxyz” Match example: Filter=”abCd”, Event Pattern=”abCd” Non-Match example: Filter=”abcd”, Event Pattern=”abcdxyz” Non-Match example: Filter=”xyz”, Event Pattern=”yz” (The entire filter does not match the last part of the event pattern;only the last two characters match.)
SA_EVT_EXACT_FILTER	The entire filter must exactly match the entire event pattern. Match example: Filter=”abc”, Event Pattern=”abc” Non-Match example: Filter=”ab”, Event Pattern=”abc” (The entire filter does not match the entire event pattern.)
SA_EVT_PASS_ALL_FILTER	Always matches, regardless of the filter or event pattern. This filter type may be used, for example, to specify a filter for event patterns 1 and 4 and a pass through for event patterns 2 and 3.

If less patterns than filters are defined, the extra filters will be matched to an empty pattern. Only a filter of size zero (*patternSize* == 0) or of type SA_EVT_PASS_ALL_FILTER matches an empty pattern.

If less filters than patterns are defined, the filter for all remaining patterns defaults to SA_EVT_PASS_ALL_FILTER. For example, if there are 10 patterns in an event and *filterCount* is 2, the first two patterns are matched against the two filters. The remaining eight patterns are automatically considered a “match”.

If the patterns of an event match the filters of several different subscriptions of a given subscriber on a single event channel, the Event Service invokes *saEvtEventDeliverCallback()* (see Section 3.6.6 on page 47) only once for that event and that subscriber. However, if a subscriber opens an event channel twice, and the patterns of an event match the filters of the subscriptions on both open event channels, the Event Service invokes *saEvtEventDeliverCallback()* twice (one for each opened channel).

3.3.7 “Lost Event” Event

A subscriber can lose events in any of the following situations:

- The subscriber handles incoming events too slowly.
- A communication failure between the subscriber and the publisher occurs.
- Communication between the subscriber and the publisher is slow.
- A node on which the subscriber or publisher is running is overloaded.

When a subscriber loses events on an event channel, the Event Service sends a **lost event** event to the subscriber on the corresponding event channel. A “lost event” notifies the subscriber that one or more events might have been lost. There is the possibility that a subscriber receives a lost event when actually the events being lost would not have matched the filters of the subscriber. The “lost event” event is delivered to the subscriber, regardless of the filters that the subscriber has set.

As soon as a process consumes a “lost event” event, the Event Service should deliver another one if one of the situations described above happens again.

The Event Service sets the attributes of the “lost event” event as follows:

- The first element of the event pattern array points to the character string defined by SA_EVT_LOST_EVENT:

  #define SA_EVT_LOST_EVENT “SA_EVT_LOST_EVENT_PATTERN”

- The event priority is set to SA_EVT_HIGHEST_PRIORITY.
- The event publish time is set to the time at which the Event Service noticed that this subscriber might have lost some events.
- The event publisher name is set to the NULL string.
- The event retention time is set to 0.
- The event identifier is set to SA_EVT_EVENTID_LOST
- The event data is empty, that is, its size is zero.

3.4 Library Life Cycle

3.4.1 saEvtInitialize()

Prototype

SaAisErrorT saEvtInitialize(

*SaEvtHandleT *evtHandle,*

*const SaEvtCallbacksT *evtCallbacks,*

*SaVersionT *version*

);

Parameters

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

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

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

Description

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

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

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

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

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

if (implementation supports the required *releaseCode*)

releaseCode = required *releaseCode*

else *{*

if (implementation supports *releaseCode* higher than the required *releaseCode*)

releaseCode = the least value of the supported release codes that is higher than the required *releaseCode*

else

releaseCode = the highest value of the supported release codes that is less than the required *releaseCode*

}

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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 Event Service library or the provider of the service is out of memory and cannot provide the service.

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

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

See Also

saEvtSelectionObjectGet(), saEvtDispatch(), saEvtFinalize()

3.4.2 saEvtSelectionObjectGet()

Prototype

```
SaAisErrorT saEvtSelectionObjectGet(
        SaEvtHandleT evtHandle,
        SaSelectionObjectT *selectionObject
);
```

Parameters

evtHandle - [in] The handle, obtained through the *saEvtInitialize()* function, designating this particular initialization of the Event Service.

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

Description

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

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

The *selectionObject* returned by *saEvtSelectionObjectGet()* is valid until *saEvtFinalize()* is invoked on the same handle *evtHandle*.

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 didn't.

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 *evtHandle* 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 Event 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).

See Also

saEvtInitialize(), saEvtDispatch(), saEvtFinalize()

3.4.3 saEvtDispatch()

Prototype

```
SaAisErrorT saEvtDispatch(
        SaEvtHandleT evtHandle,
        SaDispatchFlagsT dispatchFlags
);
```

Parameters

evtHandle - [in] The handle, obtained through the *saEvtInitialize()* function, designating this particular initialization of the Event Service.

dispatchFlags - [in] Flags that specify the callback execution behavior of the *saEvtDispatch()* function, which have the values SA_DISPATCH_ONE, SA_DISPATCH_ALL, or SA_DISPATCH_BLOCKING, as defined in volume 1 of the AIS specification.

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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 *evtHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INVALID_PARAM - The *dispatchFlags* parameter is invalid.

See Also

saEvtInitialize(), saEvtFinalize()

3.4.4 saEvtFinalize()

Prototype

```
SaAisErrorT saEvtFinalize(
        SaEvtHandleT evtHandle
);
```

Parameters

evtHandle - [in] The handle, obtained through the *saEvtInitialize()* function, designating this particular initialization of the Event Service.

Description

The *saEvtFinalize()* function closes the association, represented by the *evtHandle* parameter, between the invoking process and the Event Service. The process must

have invoked *saEvtInitialize()* before it invokes this function. A process must invoke this function once for each handle it acquired by invoking *saEvtInitialize()*.

If the *saEvtFinalize()* function returns successfully, the *saEvtFinalize()* function releases all resources acquired when *saEvtInitialize()* was called. Moreover, it closes all event channels that are open for the particular handle. Furthermore, it cancels all pending 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.

After *saEvtFinalize()* is called, the selection object is 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 didn't.

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 *evtHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

See Also

saEvtInitialize()

3.5 Event Channel Operations

3.5.1 saEvtChannelOpen() and saEvtChannelOpenAsync()

Prototype

```
SaAisErrorT saEvtChannelOpen(
        SaEvtHandleT evtHandle,
        const SaNameT *channelName,
        SaEvtChannelOpenFlagsT channelOpenFlags,
        SaTimeT timeout,
        SaEvtChannelHandleT *channelHandle
);

SaAisErrorT saEvtChannelOpenAsync(
        SaEvtHandleT evtHandle,
        SaInvocationT invocation,
        const SaNameT *channelName,
        SaEvtChannelOpenFlagsT channelOpenFlags
);
```

Parameters

evtHandle - [in] The handle, obtained through the *saEvtInitialize()* function, designating this particular initialization of the Event Service.

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

channelName - [in] A pointer to the name of the event channel that identifies an event channel globally in a cluster.

channelOpenFlags - [in] The requested access modes of the event channel. The value of this parameter is obtained by a bitwise OR of the SA_EVT_CHANNEL_PUBLISHER, SA_EVT_CHANNEL_SUBSCRIBER, and SA_EVT_CHANNEL_CREATE flags defined by *SaEvtChannelOpenFlagsT* in Section 3.3.4 on page 17. If SA_EVT_CHANNEL_PUBLISHER is set, the process may

use the returned event channel handle with *saEvtEventPublish()*.
If SA_EVT_CHANNEL_SUBSCRIBER is set, the process may use the returned event channel handle with *saEvtEventSubscribe()*. If the intent is only to open an existing event channel, the SA_EVT_CHANNEL_CREATE flag may not be set. If the intent is to open and create an event channel if it does not exist, the SA_EVT_CHANNEL_CREATE flag must be set.

timeout - [in] The *saEvtChannelOpen()* invocation is considered to have failed if it does not complete by the time specified. An event channel may still be created.

channelHandle - [out] A pointer to the handle of the event channel, provided by the invoking process in the address space of the process. If the event channel is opened successfully, the Event Service stores, in *channelHandle,* the handle that the process uses to access the channel in subsequent invocations of the functions of the Event Service API. In the case of *saEvtChannelOpenAsync((),* this handle is returned in the corresponding callback.

Description

The *saEvtChannelOpen()* and *saEvtChannelOpenAsync()* functions open an event channel. If the event channel does not exist and the SA_EVT_CHANNEL_CREATE flag is set in the *channelOpenFlags* parameter, the event channel is created first.

The *saEvtChannelOpen()* function is a blocking operation and returns a new event channel handle.

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

An event channel may be opened multiple times by the same or different processes for publishing, and subscribing to, events. If a process opens an event channel multiple times, it is possible to receive the same event multiple times. However, a process shall never receive an event more than once on a particular event channel handle.

If a process opens a channel twice and an event is matched on both open channels, the Event Service performs two callbacks -- one for each opened channel.

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 didn't.

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 *evtHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with *saEvtInitialize()* was incomplete, since the *saEvtChannelOpenCallback()* callback function is missing. This applies only to the *saEvtChannelOpenAsync()* function.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_NOT_EXIST - The event channel, identified by *channelName,* does not exist, and the value of the SA_EVT_CHANNEL_CREATE flag is not set.

SA_AIS_ERR_BAD_FLAGS - The *channelOpenFlags* parameter is invalid.

See Also

SaEvtChannelOpenCallbackT, saEvtInitialize(), saEvtChannelClose()

3.5.2 SaEvtChannelOpenCallbackT

Prototype

*typedef void (*SaEvtChannelOpenCallbackT)(*

SaInvocationT invocation,

SaEvtChannelHandleT channelHandle,

SaAisErrorT error

);

Parameters

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

channelHandle - [in] The handle that designates the event channel.

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

- SA_AIS_OK - The function completed successfully.
- SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.
- SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it didn't.
- SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may try again.
- SA_AIS_ERR_NO_MEMORY - Either the Event Service library or the provider of the service is out of memory and cannot provide the service.
- SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).
- SA_AIS_ERR_NOT_EXIST - The event channel, identified by *channelName,* does not exist, and the value of the SA_EVT_CHANNEL_CREATE flag is not set.
- SA_AIS_ERR_BAD_FLAGS - The *channelOpenFlags* parameter is invalid.

Description

The Event Service invokes this callback function when the operation requested by the invocation of *saEvtChannelOpenAsync()* completes. This callback is invoked in the context of a thread issuing an *saEvtDispatch()* call on the handle *evtHandle,* which was specified in the *saEvtChannelOpenAsync()* call.

If successful, the handle to the opened/created event channel is returned in *channelHandle;* otherwise, an error is returned in the error parameter.

Return Values

None.

See Also

saEvtChannelOpenAsync(), saEvtDispatch()

3.5.3 saEvtChannelClose()

Prototype

SaAisErrorT saEvtChannelClose(

SaEvtChannelHandleT channelHandle

);

Parameters

channelHandle - [in] The handle of the event channel to close. The *channelHandle* parameter must have been obtained previously by the invocation of one of the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions.

Description

This API function closes the event channel, designated by *channelHandle*, which was opened by an earlier invocation of the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* function.

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

When the invocation of the *saEvtChannelClose()* function completes successfully, if no process has the event channel open any longer, the event channel is deleted immediately if its deletion was pending as a result of a *saEvtChannelUnlink()* function.

Closing an event channel frees all resources allocated by the Event Service for this process, such as events allocated by the *saEvtEventAllocate()* or the *saEvtEventDeliverCallback()* functions.

The deletion of an event channel frees all resources allocated by the Event Service for it, such as published events with non-zero retention time.

If a process terminates, the Event Service implicitly closes all event channels that are open for this process.

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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 *channelHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions, or the corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

See Also

saEvtChannelOpen(), saEvtChannelOpenAsync(), SaEvtChannelOpenCallbackT, saEvtChannelUnlink()

3.5.4 saEvtChannelUnlink()

Prototype

SaAisErrorT saEvtChannelUnlink(
SaEvtHandleT evtHandle,
*const SaNameT *channelName*
);

Parameters

evtHandle - [in] The handle, obtained through the *saEvtInitialize()* function, designating this particular initialization of the Event Service.

channelName - [in] A pointer to the name of the event channel that is to be unlinked.

Description

This function deletes an existing event channel, identified by *channelName*, from the cluster.

After completion of the invocation:

- The name *channelName* is no longer valid, that is, any invocation of a function of the Event Service API that uses the event channel name returns an error, unless an event channel is re-created with this name. The event channel is re-created by specifying the same name of the event channel to be unlinked in an open call with the SA_EVT_CHANNEL_CREATE flag set. This way, a new instance of the event channel is created while the old instance of the event channel is possibly not yet finally deleted.
  Note that this is similar to the way POSIX treats files.
- If no process has the event channel open when *saEvtChannelUnlink()* is invoked, the event channel is immediately deleted.
- Any process that has the event channel open can still continue to access it. Deletion of the event channel will occur when the last *saEvtChannelClose()* operation is performed.

Note that an event channel is only deleted from the cluster namespace when *saEvtChannelUnlink()* is invoked on it.

The deletion of an event channel frees all resources allocated by the Event Service for it, such as published events with non-zero retention time.

This API can be invoked by any process, and the invoking process need not be the creator or opener of the event channel.

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 didn't.

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 *evtHandle* 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_NOT_EXIST - The event channel, identified by *channelName,* does not exist.

See Also

saEvtInitialize(), *saEvtChannelClose()*

3.6 Event Operations

3.6.1 saEvtEventAllocate()

Prototype

```
SaAisErrorT saEvtEventAllocate(
        SaEvtChannelHandleT channelHandle,
        SaEvtEventHandleT *eventHandle
);
```

Parameters

channelHandle - [in] The handle of the event channel on which the event is to be published. The *channelHandle* parameter must have been obtained previously by the

invocation of one of the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions.

eventHandle - [out] A pointer to the handle for the newly allocated event. It is the responsibility of the invoking process to allocate memory for the *eventHandle* before invoking this function. The Event Service will assign the value of the *eventHandle* when this function is invoked.

Description

The *saEvtEventAllocate()* function allocates memory for the event header, and it initializes all event attributes to default values, as described in Section 3.3.5.5 on page 19. The event allocated by *saEvtEventAllocate()* must be freed by invoking *saEvtEventFree()*.

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 didn't.

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 *channelHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions, or the corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Event 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).

See Also

saEvtEventFree(), saEvtEventPublish()

3.6.2 saEvtEventFree()

Prototype

```
SaAisErrorT saEvtEventFree(
        SaEvtEventHandleT eventHandle
);
```

Parameters

eventHandle - [in] The handle of the event whose memory can now be freed by the Event Service.

Description

The *saEvtEventFree()* function gives the Event Service permission to deallocate the memory that contains the attributes and the event data, if present, of the event that is associated with *eventHandle*. The function is used to free events allocated by *saEvtEventAllocate()* or by *saEvtEventDeliverCallback()*.

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 didn't.

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 *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained via the *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* functions, or *saEvtEventFree()* has already been invoked for *eventHandle*.

- The corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

See Also

SaEvtEventDeliverCallbackT, saEvtEventAllocate(), saEvtChannelOpen(), saEvtChannelOpenAsync()

3.6.3 saEvtEventAttributesSet()

Prototype

```
SaAisErrorT saEvtEventAttributesSet(
        SaEvtEventHandleT eventHandle,
        const SaEvtEventPatternArrayT *patternArray,
        SaEvtEventPriorityT priority,
        SaTimeT retentionTime,
        const SaNameT *publisherName
);
```

Parameters

eventHandle - [in] The handle of the event whose attributes are to be set.

patternArray - [in] A pointer to a structure that contains the array of patterns to be copied into the event pattern array and the number of such patterns.

priority - [in] The priority of the event.

retentionTime - [in] The duration for which the event will be retained.

publisherName - [in] A pointer to the name of the publisher of the event.

Description

This function is used to set all the writeable event attributes, that is, *patternArray*, *priority*, *retentionTime,* and *publisherName*, in the header of the event, designated by the *eventHandle* handle. If *patternArray* or *publisherName* are NULL, the corresponding attributes are not changed.

Once the call to *saEvtEventAttributesSet()* returns, *patternArray* can be freed.

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 didn't.

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 *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained via the *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* functions, or *saEvtEventFree()* has already been invoked for *eventHandle*.
- The corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_ACCESS - The access to this event is denied. In particular, a subscriber is not allowed to modify an event delivered by the *saEvtEventDeliverCallback()* callback function.

SA_AIS_ERR_TOO_BIG - the value *patternSize* of one or more patterns or the value *patternsNumber* specified in the *patternArray* parameter is larger than the maximum value permitted.

See Also

saEvtEventAllocate(), saEvtEventFree(), SaEvtEventDeliverCallbackT, saEvtEventAttributesGet(), saEvtChannelOpen(), *saEvtChannelOpenAsync()*

3.6.4 saEvtEventAttributesGet()

Prototype

SaAisErrorT saEvtEventAttributesGet(
SaEvtEventHandleT eventHandle,
*SaEvtEventPatternArrayT *patternArray,*
*SaEvtEventPriorityT *priority,*
*SaTimeT *retentionTime,*
*SaNameT *publisherName,*
*SaTimeT *publishTime,*
*SaEvtEventIdT *eventId*
);

Parameters

eventHandle - [in] The handle of the event whose attributes are to be retrieved.

patternArray - [in/out] A pointer to a structure that describes the event pattern array and the number of patterns to be retrieved.

If the caller sets *patternArray->patterns* to NULL, the Event Service ignores the *patternArray->allocatedNumber* field, and it will allocate memory for the pattern array and the individual patterns and will set the fields *patternArray->patternsNumber, patternArray->patterns, patternArray->patterns[i].pattern,* and *patternArray->patterns[i].patternSize* accordingly. The invoking process is then responsible for de-allocating the corresponding memory, that is, *patternArray->patterns* and each *patternArray->patterns[i].pattern.*

Alternatively, the invoking process can allocate the memory to retrieve all event patterns and set the fields *patternArray->allocatedNumber, patternArray->patterns, patternArray->patterns[i].allocatedSize,* and *patternArray->patterns[i].pattern* accordingly. In this case, these fields are in parameters and will not be modified by the Event Service.
The Event Service copies the patterns into the successive entries of *patternArray->patterns,* starting with the first entry and continuing until all event patterns are copied. If *patternArray->allocatedNumber* is smaller than the number of event patterns or if the size of the buffer allocated for one of the patterns is smaller

than the actual size of the pattern, the invocation fails and the SA_AIS_ERR_NO_SPACE error is returned. If such an error happens, it is unspecified whether some buffers pointed to by *patternArray->patterns[i].pattern* were changed or not by the Event Service. Regardless of whether such an error occurs, the Event Service sets the *patternArray->patternsNumber* and *patternArray->patterns[i].patternSize* fields for all *patternArray->allocatedNumber* individual patterns, to indicate the number of event patterns and the size of each pattern.

priority - [out] A pointer to the priority of the event.

retentionTime - [out] A pointer to the duration for which the publisher will retain the event.

publisherName - [out] A pointer to the name of the publisher of the event.

publishTime - [out] A pointer to the time at which the publisher published the event.

eventId - [out] A pointer to the event identifier.

Description

This function retrieves the value of the attributes of the event designated by *eventHandle*.

For each of the out or in/out parameters, if the invoking process provides a NULL reference, the Event Service does not return the out value.

It is possible to invoke this API function on any event allocated by the *saEvtEventAllocate()* function or received via the *saEvtEventDeliverCallback()* function, and possibly modified by the *saEvtEventAttributesSet()* function. Only if this API is invoked on a received event, the attributes

- "publish time" and
- "event id"

have the values set by the Event Service at event publishing time. In all other cases, the attributes will either have the initial values set by the Event Service when allocating the event or the attributes set by a prior invocation of the *saEvtEventAttributesSet()* function.

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 didn't.

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 *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained via the *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* functions, or *saEvtEventFree()* has already been invoked for *eventHandle*.
- The corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_SPACE - The field *allocatedNumber* in *patternArray* is smaller than the number of event patterns or the size of the buffer allocated for one of the patterns is smaller than the actual size of the pattern. This return value applies only if the patterns field in patternArray as an in parameter is not NULL.

See Also

SaEvtEventDeliverCallbackT, saEvtEventAllocate(), saEvtEventFree(), saEvtChannelOpen(), saEvtChannelOpenAsync(), saEvtEventAttributesSet()

3.6.5 saEvtEventDataGet()

Prototype

SaAisErrorT saEvtEventDataGet(

SaEvtEventHandleT eventHandle,

*void *eventData,*

*SaSizeT *eventDataSize*

);

Parameters

eventHandle - [in] The handle to the event previously delivered by *saEvtEventDeliverCallback()*.

eventData - [in/out] A pointer to a buffer provided by the process in which the Event Service stores the data associated with the delivered event.
If *eventData* is NULL, the value of *eventDataSize* provided by the invoking process is ignored, and the buffer is provided by the Event Service library. The buffer must be deallocated by the calling process after returning from the *saEvtEventDataGet()* call.

eventDataSize - [in/out] If *eventData* is not NULL, the in value of *eventDataSize* is the size of the *eventData* buffer provided by the invoking process. If this buffer is not large enough to hold all of the data associated with this event, then no data will be copied into the buffer, and the error code SA_AIS_ERR_NO_SPACE is returned.
If *eventData* is NULL, the in value of *eventDataSize* is ignored.
The out value of *eventDataSize* is set when the function returns either SA_AIS_OK or SA_AIS_ERR_NO_SPACE, and it is the size of the data associated with this event, which may be less than, equal to, or greater than the in value of *eventDataSize*.

Description

The *saEvtEventDataGet()* function allows a process to retrieve the data associated with an event previously delivered by *saEvtEventDeliverCallback()*.

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 didn't.

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 *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained via the *saEvtEventDeliverCallback()* function, or *saEvtEventFree()* has already been invoked for *eventHandle*.
- The corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_NO_SPACE - The buffer provided by the process is too small to hold the data associated with the delivered event.

See Also

SaEvtEventDeliverCallbackT, saEvtEventFree(), saEvtChannelOpen(), saEvtChannelOpenAsync()

3.6.6 SaEvtEventDeliverCallbackT

Prototype

*typedef void(*SaEvtEventDeliverCallbackT)(*

SaEvtSubscriptionIdT subscriptionId,

SaEvtEventHandleT eventHandle,

SaSizeT eventDataSize

);

Parameters

subscriptionId - [in] An identifier that a process supplied in an *saEvtEventSubscribe()* invocation that enables it to determine which subscription resulted in the delivery of the event.

eventHandle - [in] The handle to the event delivered by this callback.

eventDataSize - [in] The size of the data associated with the event.

Description

The Event Service invokes this callback function to notify a subscribing process that an event has been received. This callback is invoked in the context of a thread issuing an *saEvtDispatch()* call on the handle *evtHandle,* which was specified in the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* calls, leading to the handle *eventHandle.*

A published event is received when it has an event pattern matching the filter of a subscription of this process, established by the *saEvtEventSubscribe()* call. For details on filtering, refer to Section 3.3.6 on page 20. After successful completion of this call, the process may invoke the *saEvtEventAttributesGet()* function to obtain the attributes associated with the event and the *saEvtEventDataGet()* function to obtain the data associated with the event.

It is the responsibility of the process to free the event by invoking the s*aEvtEventFree()* function.

The validity of the *eventHandle* parameter is not limited to the scope of this callback function.

Return Values

None.

See Also

saEvtEventAttributesGet(), saEvtEventDataGet(), saEvtEventSubscribe(), saEvtChannelOpen(), saEvtChannelOpenAsync(), saEvtEventFree(), saEvtDispatch()

3.6.7 saEvtEventPublish()

Prototype

```
SaAisErrorT saEvtEventPublish(
        SaEvtEventHandleT eventHandle,
        const void *eventData,
        SaSizeT eventDataSize,
        SaEvtEventIdT *eventId
);
```

Parameters

eventHandle - [in] The handle of the event that is to be published. The event must have been allocated by the *saEvtEventAllocate()* function or obtained via the *saEvtEventDeliverCallback()* function, and the patterns must have been set by *saEvtEventAttributesSet(),* if changes are required.

eventData - [in] A pointer to a buffer that contains additional event information for the event being published. This parameter is set to NULL if no additional information is associated with the event. The process may deallocate the memory for *eventData* when *saEvtEventPublish()* returns.

eventDataSize - [in] The number of bytes in the buffer pointed to by *eventData*. This parameter is ignored if *eventData* is NULL.

eventId - [out] A pointer to an identifier of the event.

Description

The *saEvtEventPublish()* function publishes an event on the channel for which the event specified by *eventHandle* has been allocated or obtained via the *saEvtEventDeliverCallback()* function, and returns the event identifier in *eventId.* The event to be published consists of a standard set of attributes (the event header) and an optional data part.

The process must have opened the event channel on which this event is published with the SA_EVT_CHANNEL_PUBLISHER flag set for an invocation of this function to be successful.

Before an event can be published, the publisher process can invoke the *saEvtEventAttributesSet()* function to set the writeable event attributes. The published event is delivered to subscribers whose subscription filters match the event patterns.

When the Event Service publishes an event, it automatically sets the following read-only event attributes into the published event:

- Event publish time
- Event identifier

In addition to the event attributes, a process can supply values for the *eventData* and *eventDataSize* parameters for publication as part of the event.

The event attributes and the event data of the event identified by *eventHandle* are not affected by this API function.

The invocation of *saEvtEventPublish()* copies the event attributes and the event data into internal memory of the Event Service. The invoking process can free the event using *saEvtEventFree()* after *saEvtEventPublish()* returns.

Return Values

SA_AIS_OK - The function call 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 didn’t.

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 *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained via the *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* functions, or *saEvtEventFree()* has already been invoked for *eventHandle*.
- The corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_PUBLISHER flag was not set for the event channel on which the event to be published was allocated, i.e., the event channel was not opened for publisher access.

SA_AIS_ERR_TOO_BIG - The *eventDataSize* or the total size of the event is larger than the maximum permitted value.

See Also

saEvtEventAttributesSet(), saEvtEventSubscribe(), saEvtEventAllocate(), saEvtEventFree(), SaEvtEventDeliverCallbackT

3.6.8 saEvtEventSubscribe()

Prototype

```
SaAisErrorT saEvtEventSubscribe(
        SaEvtChannelHandleT channelHandle,
        const SaEvtEventFilterArrayT *filters,
        SaEvtSubscriptionIdT subscriptionId
);
```

Parameters

channelHandle - [in] The handle of the event channel on which the process is subscribing to receive events. The parameter *channelHandle* must have been obtained previously by the invocation of one of the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions.

filters - [in] A pointer to a *SaEvtEventFilterArrayT* structure, allocated by the process, that defines filter patterns to use to filter events received on the event channel. The process may deallocate the memory for the filters when *saEvtEventSubscribe()* returns.

subscriptionId - [in] An identifier that uniquely identifies a specific subscription on this instance of the opened event channel corresponding to the *channelHandle* and that is used as a parameter of *saEvtEventDeliverCallback*().

Description

The *saEvtEventSubscribe()* function enables a process to subscribe for events on an event channel by registering one or more filters on that event channel.

Events are delivered via the invocation of the *saEvtEventDeliverCallback()* callback function, which must have been supplied when the process called the *saEvtInitialize()* function.
The process must have opened the event channel, designated by *channelHandle,* with the SA_EVT_CHANNEL_SUBSCRIBER flag set for an invocation of this function to be successful.

The memory associated with the filters is not deallocated by the *saEvtEventSubscribe()* function. It is the responsibility of the invoking process to de-allocate the memory when the *saEvtEventSubscribe()* function returns.

For a given subscription, the filters parameter cannot be modified. To change the filters parameter without losing events, a process must establish a new subscription with the new filters parameter. After the new subscription is established, the old subscription can be removed by invoking the *saEvtEventUnsubscribe()* function.

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 didn't.

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 *channelHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions, or the corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with *saEvtInitialize()* was incomplete, since the *saEvtEventDeliverCallback()* callback function is missing.

SA_AIS_ERR_INVALID_PARAM - The *filters* parameter is invalid.

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

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

SA_AIS_ERR_EXIST - A subscription using the same *subscriptionId* already exists.

SA_AIS_ERR_ACCESS - The channel, identified by *channelHandle,* was not opened with the SA_EVT_CHANNEL_SUBSCRIBER flag set, i.e., the channel was not opened for subscriber access.

SA_AIS_ERR_TOO_BIG - The field *filtersNumber* in *filters* or the length of one or more filter strings exceeds the maximum permitted size.

See Also

SaEvtEventDeliverCallbackT, saEvtEventUnsubscribe(), saEvtEventDataGet(), saEvtEventAttributesGet()

3.6.9 saEvtEventUnsubscribe()

Prototype

SaAisErrorT saEvtEventUnsubscribe(

SaEvtChannelHandleT channelHandle,

SaEvtSubscriptionIdT subscriptionId

);

Parameters

channelHandle - [in] The event channel for which the subscriber is requesting the Event Service to delete the subscription. The parameter *channelHandle* must have

been obtained previously by the invocation of one of the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions.

subscriptionId - [in] The identifier of the subscription that the subscriber is requesting the Event Service to delete.

Description

The *saEvtEventUnsubscribe()* function allows a process to stop receiving events for a particular subscription on an event channel by removing the subscription.

The *saEvtEventUnsubscribe()* operation is successful if the *subscriptionId* parameter matches a previously registered subscription. Events queued to be delivered to the process, and that no longer match any subscription, because the *saEvtEventUnsubscribe()* operation has been invoked, are purged. A process that wishes to modify a subscription without losing any events must establish the new subscription before removing the existing subscription.

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 didn't.

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 *channelHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions, or the corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_NOT_EXIST - The s*ubscriptionId* parameter does not match any currently registered subscription for the calling process.

See Also

saEvtEventSubscribe()

3.6.10 saEvtEventRetentionTimeClear()

Prototype

SaAisErrorT saEvtEventRetentionTimeClear(

SaEvtChannelHandleT channelHandle,

const SaEvtEventIdT eventId

);

Parameters

channelHandle - [in] The handle of the event channel on which the event has been published. The handle c*hannelHandle* must have been obtained previously by the invocation of one of the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions.

eventId - [in] The identifier of the event.

Description

The *saEvtEventRetentionTimeClear()* function is used to clear the retention time of a published event, designated by *eventId*. This function indicates to the Event Service that the Event Service does not need to keep the event any longer for potential new subscribers. Once the value of the retention time is reset to 0, the event is no longer available for new subscribers.

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 didn't.

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 *channelHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saEvtChannelOpen()* or *saEvtChannelOpenCallback()* functions, or the corresponding event channel has already been closed.
- The handle *evtHandle* that was passed to the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this error is returned if *eventId* is not a valid event identifier.

SA_AIS_ERR_NOT_EXIST - The event specified by *eventId* does not exist.

See Also

saEvtEventPublish(), SaEvtEventDeliverCallbackT