Service Availability^{TM} Forum Application Interface Specification

Event Service SAI-AIS-EVT-B.03.01

SERVICE AVAILABILITY™ FORUM

.

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

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

1.2 AIS Documents Organization

The Application Interface Specification is organized into several volumes. For a list of all Application Interface Specification documents, refer to the SA Forum Overview document ([1]).

1.3 History

Previous releases of the Event Service specification:

(1) SAI-AIS-EVT-A.01.01
(2) SAI-AIS-EVT-B.01.01
(3) SAI-AIS-EVT-B.02.01

This section presents the changes of the current release, SAI-AIS-EVT-B.03.01, with respect to the SAI-AIS-EVT-B.02.01 release. Editorial changes are not mentioned here.

1.3.1 New Topics

- Section 3.2 on page 16 describes the behavior of the Event Service API on a cluster node that is not in the cluster membership (see [4]).
- Section 3.4.8 on page 27 describes the *SaEvtLimitIdT* enum, which provides a set of values that identify limits for a particular implementation of the Event Service. The user can inquire at runtime the current value of a particular limit by specifying the corresponding enum value when invoking the *saEvtLimitGet()* function defined in Section 3.8.1 on page 66.

- Chapter 4 presents the Event Service UML Information Model. The Event Service UML classes (see FIGURE 1 on page 70) were previously contained in [1].
- Chapter 5 states that no administration APIs are provided for the Event Service.
- Chapter 7 presents the Event Service Management Interface.

1.3.2 Clarifications

- Section 3.4.7 on page 25 clarifies when the Event Service sends “lost event” to a subscriber.
- Section 3.5.3 on page 32 on the *saEvtDispatch()* function clarifies the meaning of the SA_AIS_OK return value.
- The description of the *saEvtFinalize()* function (see Section 3.5.4 on page 33) clarifies that this function frees all resources allocated by the Event Service for the process in this association between the process and the Event Service.
- The description of the *saEvtChannelClose()* function (see Section 3.6.3 on page 40) clarifies which resources this function frees for the invoking process.
- The *saEvtEventFree()* function (see Section 3.7.2 on page 46) clarifies when the data pertaining to the event is freed.

1.3.3 Changes in Return Values of API Functions:

Table 1 Changes in Return Values of API Functions

API Function	Return Value	Change Type
All API functions except *saEvtFinalize()* and *SaEvtEventDeliverCallbackT*	SA_AIS_ERR_UNAVAILABLE	new
saEvtChannelOpen(), *saEvtChannelOpenAsync()*, and *SaEvtChannelOpenCallbackT*	SA_AIS_ERR_NO_RESOURCES	clarified
saEvtEventAttributesSet()	SA_AIS_ERR_TOO_BIG	extended
saEvtEventPublish() and *saEvtEventSubscribe()*	SA_AIS_ERR_TOO_BIG	clarified
saEvtEventSubscribe()	SA_AIS_ERR_INVALID_PARAM	extended

1.3.4 Removed Topics

SA Forum revisited its alarm issuance directives for this release and modified the conditions that determine when an alarm would be produced. As a consequence, AIS services shall only generate alarms for situations that require an explicit intervention by an external agent or operator, provided that the corrective measures to be taken are well defined. Based on these directives, the alarms generated so far by the AIS services have been revised, and it was decided to remove all alarms from the Event Service B.03.01 version.

SA Forum does not mandate that Event Service implementations which also support the B.02.01 version must generate the now removed alarms for the B.02.01 version.

These alarms have also been removed from the Event Service MIB for the Event Service B.03.01 version.

1.3.5 Other Changes

In Section 3.4.5.4 on page 21 on the *SaEvtEventIdT* type, “LL” has been appended to the numeric values of constants.

1.4 References

The following documents contain information that is relevant to the specification:

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

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

[3] Service Availability^{TM} Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.02.01

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

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

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

References to these documents are made by putting the number of the document in brackets.

1.5 How to Provide Feedback on the Specification

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

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

1.6 How to Join the Service Availability™ Forum

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

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

1.7 Additional Information

1.7.1 Member Companies

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

1.7.2 Press Materials

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

2 Overview

This specification defines the 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. One or more publishers communicate asynchronously with one or more subscribers by means of events over a cluster-wide entity named 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 called the **event header**) and zero or more bytes of **event data**.

A process invokes the *saEvtEventAllocate()* function to allocate the event header and the *saEvtEventFree()* function to deallocate it. The *saEvtEventAllocate()* function returns a handle that can be used in subsequent invocations of functions provided by the Event Service API.

A process writes the event attributes by invoking the *saEvtEventAttributesSet()* function and reads these attributes by invoking the *saEvtEventAttributesGet()* function.

An event is published by invoking the *saEvtEventPublish()* function and specifying as parameters the event handle and optional additional information, the event data, which is contained in a separate free-form data buffer. A **published event** is an event that has been handed over to the Event Service by successfully invoking the *saEvtEventPublish()* function and consists of the event header, which contains the set of attributes, and optional additional information, the event data. A published event is owned by the Event Service. This document uses in some places simply "event" instead of "published event " when the context makes clear what is meant.

3.1.2 Event Channels

An **event channel** is global to a cluster and is identified by a unique name. An event channel enables multiple publishers to communicate with multiple subscribers. A **publisher** on an event channel is a process that successfully opened the event channel with the SA_EVT_CHANNEL_PUBLISHER set. A **subscriber** on an event channel is a process that successfully called the *saEvtEventSubscribe()* function to subscribe for events on this event channel. A process opens an event channel by invoking one of the *saEvtEventChannelOpen()* or *saEvtEventChannelOpenAsync()* functions. The process can specify in the open call whether it wants to access only 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 by invoking the *saEvtChannelUnlink()* function.

Once an event has been allocated for an event channel by invoking the *saEvtEventAllocate()* function, 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, it is not guaranteed 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[1] 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 particular priority level, events sent by a publisher are received by subscribers in the order in which the publisher published the events.
- **Event completeness** - A published event is complete in the sense that it contains the entire event data provided by the caller (if any) and all event attributes provided by the caller or supplied by the Event Service when the event was published. The Event Service guarantees that a process subscribing for events either obtains complete events or no event at all.
- **Retention time** - Events published with a nonzero retention time are kept for the specified duration. This gives new subscribers the opportunity 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:

1. The term “node” without a preceding qualifier “cluster” or “member” is used in this document in the sense of a “member node”, as defined in the Cluster Membership Service specification (see [4]).

- **Persistence** - Published events may be persisted and may survive node failures or a shut down (in the operating system sense) of the entire cluster, but that is not mandated 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. The Event Service delivers events to a subscribing process by invoking the *saEvtEventDeliverCallback()* function of that process. To stop receiving events for which a subscriber has registered, the 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 total size of the event is larger than the maximum value supported by the implementation.".

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 that use only three patterns 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 a process subscribes on an event channel to receive published events, it must specify which filters to apply on the published events. Events are only delivered to a process if they satisfy the provided filters. For a description of the filtering process, refer to Section 3.4.6.3 on page 23.

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 in which the subscriber is interested.
- The name of a table in which the subscriber is interested.
- 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 Unavailability of the Event Service API on a Non-Member Node

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

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

3.2.1 A Member Node Leaves or Rejoins the Cluster Membership

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

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

⇒ All Event Service APIs that are invoked by the process and that operate on handles already acquired by the process will fail with SA_AIS_ERR_UNAVAILABLE with the following exceptions, assuming that the handle *evtHandle* has already been acquired:

- The *saEvtChannelOpenAsync()* function may return SA_AIS_OK or SA_AIS_ERR_UNAVAILABLE, depending on the service implementation. If it returns SA_AIS_OK, the callback *SaEvtChannelOpenCallbackT* will be called and will also return SA_AIS_ERR_UNAVAILABLE in the *error* parameter; otherwise, the callback will not be called.
- The *saEvtFinalize()* function, which is used to free the library handles and all resources associated with these handles.

⇒ An outstanding callback *SaEvtChannelOpenCallbackT* will return SA_AIS_ERR_UNAVAILABLE in the *error* parameter.

⇒ The callback *SaEvtEventDeliverCallbackT* will not be called.

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

When the node leaves the membership, the Event Service executing on the remaining nodes of the cluster behaves as if all processes that were using the Event Service on the leaving node had been terminated. In particular, if an *saEvtChannelUnlink()* operation is pending because one or more processes on the leaving node had the event channel open, the unlink operation can proceed now.

3.2.2 Guidelines for Event Service Implementers

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

3.3 Include File and Library 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

3.4 Type Definitions

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

3.4.1 Handles

3.4.1.1 SaEvtHandleT

typedef SaUint64T SaEvtHandleT;

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

3.4.1.2 SaEvtEventHandleT

typedef SaUint64T SaEvtEventHandleT;

This type is used for the handle to an event.

3.4.1.3 SaEvtChannelHandleT

typedef SaUint64T SaEvtChannelHandleT;

This type is used for the handle to an open event channel.

3.4.2 SaEvtSubscriptionIdT

typedef SaUint32T SaEvtSubscriptionIdT;

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

3.4.3 SaEvtCallbacksT

The *SaEvtCallbacksT* type is defined as follows:

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

A structure of the *SaEvtCallbacksT* type (called a callbacks structure) is used to specify the callback functions that the Event Service can invoke.

3.4.4 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 by invoking either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()*, some combination of these flags are bitwise ORed together to provide the value of the *channelOpenFlags*.

3.4.5 Event Patterns and Attributes

The *SaEvtEventPatternT* type is defined below. An **event pattern** may contain a name (for instance, process name, checkpoint name, service instance name, and so on). Alternatively, an event pattern may characterize an event (for instance, timedOut, newComponent, overload, and so on).

3.4.5.1 SaEvtEventPatternT

```
typedef struct {
        SaSizeT allocatedSize;
        SaSizeT patternSize;
        SaUint8T *pattern;
} SaEvtEventPatternT;
```

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

- *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 to which the pattern value will be copied

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

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

3.4.5.2 SaEvtEventPatternArrayT

```
typedef struct {
        SaSizeT allocatedNumber;
        SaSizeT patternsNumber;
        SaEvtEventPatternT *patterns;
} SaEvtEventPatternArrayT;
```

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

- *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 to which the array of patterns will be copied

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

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

3.4.5.3 SaEvtEventPriorityT

#define SA_EVT_HIGHEST_PRIORITY 0

#define SA_EVT_LOWEST_PRIORITY 3

typedef SaUint8T SaEvtEventPriorityT;

3.4.5.4 SaEvtEventIdT

typedef SaUint64T SaEvtEventIdT;

This type is used for 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 0LL

Event identifier for an allocated but not yet published event.

#define SA_EVT_EVENTID_LOST 1LL

Event identifier for a “lost event”.

3.4.5.5 Event Attributes

A process has read access to all attributes of an event allocated by *saEvtEventAllocate()* or obtained from *saEvtEventDeliverCallback()* by specifying the event handle returned by these functions in a call to *saEvtEventAttributesGet()*.

A process may not access the event attributes of a published event (which is owned by the Event Service), except when discarding a published event with nonzero retention time by invoking the *saEvtEventRetentionTimeClear()* function to clear the retention time of the event. For this purpose, the process uses the event id that it obtained in a previous *saEvtEventPublish()* call to refer to the published event.

The following list shows all event attributes. For each attribute, it is specified whether the process in which the event resides has write access or not, as some attributes can be set only by the Event Service. Additionally, the list gives the default value of each attribute of an event allocated by invoking the *saEvtEventAllocate()* function.

- **Event Pattern Array** - An array defined earlier 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. This time 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 attribute is read-only.
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. If the publishing process is part of a component under the control of the Availability Management Framework, this field should contain the name of that component (in future, it is expected that in such cases it shall be mandatory to pass the LDAP DN of a component); otherwise, any octet string (including zeros) may be used as the name.
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 attribute is read-only.
Default: SA_EVT_EVENTID_NONE

3.4.6 Event Filters

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

3.4.6.1 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 *saEvtEventFilterTypeT* enumeration type defines the types of filters.
The corresponding pattern matching algorithms are explained later in Table 2 on page 24.

3.4.6.2 SaEvtEventFilterT

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

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

3.4.6.3 SaEvtEventFilterArrayT

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

The event filter array structure type defines one or more filters.

Filters are passed to the Event Service by a subscriber process by invoking the *saEvtEventSubscribe()* function. The Event Service does the filtering to decide whether a published event is delivered to a subscriber for a particular 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 particular subscription if all patterns of the event match all filters provided in an invocation of the *saEvtEventSubscribe()* call.

Table 2 Filter Types and Pattern Matching Algorithms

Filter Type	Matching Algorithm
SA_EVT_PREFIX_FILTER	The entire filter must match the first *filter.patternSize* 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 *filter.patternSize* 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 fewer patterns than filters are defined, the extra filters will be matched to an empty pattern. Only a filter of size zero (*filter.patternSize* == 0) or of type SA_EVT_PASS_ALL_FILTER matches an empty pattern.

If fewer filters than patterns are defined, the filter for all remaining patterns defaults to SA_EVT_PASS_ALL_FILTER. For example, if an event has 10 patterns and *filtersNumber* 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 particular subscriber on a single event channel, the Event Service invokes *saEvtEventDeliverCallback()* (see Section 3.7.7 on page 56) 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).

3.4.7 “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. It is possible 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.

Regardless of the situations mentioned above that can cause a subscriber to lose events, an Event Service implementation must make sure that if a publisher sends events A, B, and C in this order, and the Event Service determines that it can deliver A and C but not B to a subscriber, it should deliver a "lost event" between A and C.

As soon as a process consumes a “lost event” event, the Event Service should deliver another one if one or more of the situations previously described occur 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.

3.4.8 SaEvtLimitIdT

The *SaEvtLimitIdT* enum provides a set of values that identify limits for a particular implementation of the Event Service. Note that the Event Service specification does not define a configuration for these limits, which are usually predefined by the implementation.

The user can retrieve at runtime the current value of a particular limit by specifying the identifier of the limit (one of the enum values of the type *SaEvtLimitIdT*, defined below) when invoking the *saEvtLimitGet()* function (see Section 3.8.1 on page 66).

The limit value is returned in a parameter of a generic type (*SaLimitValueT* type, defined in [1]). As the limit identified by the enum SA_EVT_MAX_RETENTION_DURATION_ID is of type *SaTimeT*, the *timeValue* field of *SaLimitValueT* must be used for further access. As all other limits defined in this specification are of type *SaUint64T*, the *uint64Value* field of *SaLimitValueT* must be used for further access.

```
typedef enum {
        SA_EVT_MAX_NUM_CHANNELS_ID          = 1,
        SA_EVT_MAX_EVT_SIZE_ID              = 2,
        SA_EVT_MAX_PATTERN_SIZE_ID          = 3,
        SA_EVT_MAX_NUM_PATTERNS_ID          = 4,
        SA_EVT_MAX_RETENTION_DURATION_ID    = 5,
} SaEvtLimitIdT;
```

The values of the *SaEvtLimitIdT* enumeration type have the following interpretation:

- SA_EVT_MAX_NUM_CHANNELS_ID - This enum can be used to retrieve the maximum number of event channels in the cluster.
- SA_EVT_MAX_EVT_SIZE_ID - This enum can be used to retrieve the maximum size in bytes of the event. This size includes the size of the header and the size of the data.
- SA_EVT_MAX_PATTERN_SIZE_ID - This enum can be used to retrieve the maximum size in bytes of a pattern of an event.
- SA_EVT_MAX_NUM_PATTERNS_ID - This enum can be used to retrieve the maximum number of patterns an event can have. Note that the maximum number of filters that can be specified is the same as the maximum number of patterns an event can have.
- SA_EVT_MAX_RETENTION_DURATION_ID - This enum can be used to retrieve the longest period an event will be retained.

3.5 Library Life Cycle

3.5.1 saEvtInitialize()

Prototype

```
SaAisErrorT saEvtInitialize(
        SaEvtHandleT *evtHandle,
        const SaEvtCallbacksT *evtCallbacks,
        SaVersionT *version
);
```

Parameters

evtHandle - [*out*] A pointer to the handle which designates this particular initialization of the Event Service, and which is to be returned by the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

evtCallbacks - [*in*] If *evtCallbacks* is set to NULL, no callback is registered; If *evtCallbacks* is not set to NULL, it is a pointer to an *SaEvtCallbacksT* structure which contains the callback functions of the process that the Event Service may invoke. Only non-NULL callback functions in this structure will be registered. The *SaEvtCallbacksT* type is defined in Section 3.4.3 on page 18.

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

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 pointed to by *evtHandle* is returned by the Event Service 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 specified *releaseCode* and *majorVersion*, SA_AIS_OK is returned. In this case, the structure pointed to by the *version* parameter is set by this function to:

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

If the preceding condition cannot be met, SA_AIS_ERR_VERSION is returned, and the structure pointed to by the *version* parameter is set to:

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

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the 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 provided in the structure to which the *version* parameter points is not compatible with the version of the Event Service implementation.

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

See Also

saEvtSelectionObjectGet(), *saEvtDispatch()*, *saEvtFinalize()*

3.5.2 saEvtSelectionObjectGet()

Prototype

SaAisErrorT saEvtSelectionObjectGet(

SaEvtHandleT evtHandle,

*SaSelectionObjectT *selectionObject*

);

Parameters

evtHandle - [*in*] The handle which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

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

Description

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

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

The operating system handle returned by *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 did not.

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

SA_AIS_ERR_BAD_HANDLE - The handle *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).

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

See Also

saEvtInitialize(), *saEvtDispatch()*, *saEvtFinalize()*

3.5.3 saEvtDispatch()

Prototype

SaAisErrorT saEvtDispatch(

SaEvtHandleT evtHandle,

SaDispatchFlagsT dispatchFlags

);

Parameters

evtHandle - [*in*] The handle which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

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

Description

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

Return Values

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

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - The *dispatchFlags* parameter is invalid.

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

See Also

saEvtInitialize(), *saEvtFinalize()*

3.5.4 saEvtFinalize()

Prototype

SaAisErrorT saEvtFinalize(

SaEvtHandleT evtHandle

);

Parameters

evtHandle - [*in*] The handle which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

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 completes successfully, it releases all resources acquired when the *saEvtInitialize()* function was called. Moreover, it closes all event channels that are open for the particular handle. Furthermore, it cancels all pending *SaEvtChannelOpenCallbackT* callbacks related to the particular handle. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

See Also

saEvtInitialize(), *saEvtChannelClose()*, *SaEvtChannelOpenCallbackT*, *saEvtSelectionObjectGet()*

3.6 Event Channel Operations

3.6.1 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 which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

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

channelName - [*in*] A pointer to the name of the event channel that identifies an event channel globally in a cluster. The *SaNameT* type is defined in [1].

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, as defined for the *SaEvtChannelOpenFlagsT* type in Section 3.4.4 on page 19. If SA_EVT_CHANNEL_PUBLISHER is set, the process may use the returned event channel handle when invoking *saEvtEventPublish()*. If SA_EVT_CHANNEL_SUBSCRIBER is set, the process may use the returned event channel handle when invoking *saEvtEventSubscribe()*. If the user intends to open only an existing event channel, the SA_EVT_CHANNEL_CREATE flag may not be set. If the user intends to open an event channel or create and open 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. The *SaTimeT* type is defined in [1].

channelHandle - [*out*] A pointer to the memory area (provided by the invoking process in the address space of the process) to hold the channel handle. If the event channel is opened successfully, the Event Service stores in this memory area 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. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

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 first created.

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 events and subscribing to events. If a process opens an event channel multiple times, it may 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 did not.

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

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

SA_AIS_ERR_INIT - The previous invocation of *saEvtInitialize()* to initialize the Event Service 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. In particular, this value is returned if SA_EVT_CHANNEL_CREATE is set in *channelOpenFlags*, and the name to which *channelName* points is not a DN, or the type of its first RDN is not *safChnl*.

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). In particular, this value is returned if the number of event channels in the cluster has reached its limit, and the call would create a new event channel. Refer to the description of the enum SA_EVT_MAX_NUM_CHANNELS_ID in Section 3.4.8 on page 27.

SA_AIS_ERR_NOT_EXIST - The SA_EVT_CHANNEL_CREATE flag is not set and the event channel designated by the name to which *channelName* points does not exist.

SA_AIS_ERR_BAD_FLAGS - The *channelOpenFlags* parameter is invalid.

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

See Also

SaEvtChannelOpenCallbackT, *saEvtInitialize()*, *saEvtChannelClose()*

3.6.2 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. The *invocation* parameter allows the process to match the invocation of that function with this callback. The *SaInvocationT* type is defined in [1].

channelHandle - [*in*] The handle that designates the event channel. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

error - [*in*] This parameter indicates whether the *saEvtChannelOpenAsync()* function was successful. The *SaAisErrorT* type is defined in [1]. The values that can be returned are:

- SA_AIS_OK - The function completed successfully.
- SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.
- SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.
- SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may try again.
- SA_AIS_ERR_BAD_HANDLE - The handle *evtHandle* in the corresponding invocation of the *saEvtChannelOpenAsync()* function is invalid, since it is corrupted, uninitialized, or has already been finalized.
- SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly in the corresponding invocation of the *saEvtChannelOpenAsync()* function. In particular, this value is returned if SA_EVT_CHANNEL_CREATE is set in *channelOpenFlags*, and the name to which *channelName* points is not a DN, or the type of its first RDN is not *safChnl*.

- 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). In particular, this value is returned if the number of event channels in the cluster has reached its limit, and the call would create a new event channel. Refer to the description of the enum SA_EVT_MAX_NUM_CHANNELS_ID in Section 3.4.8 on page 27.
- SA_AIS_ERR_NOT_EXIST - In the corresponding invocation of the *saEvtChannelOpenAsync()* function, the SA_EVT_CHANNEL_CREATE flag is not set, and the event channel designated by the name to which *channelName* points does not exist.
- SA_AIS_ERR_BAD_FLAGS - The *channelOpenFlags* parameter in the corresponding invocation of the *saEvtChannelOpenAsync()* function is invalid.
- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

Description

The 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 calling *saEvtDispatch()* on the handle *evtHandle* that was specified in the *saEvtChannelOpenAsync()* call.

If this function succeeds, 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()*

3.6.3 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 in an invocation of *saEvtChannelOpen()* or *saEvtChannelOpenCallback()*. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

Description

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

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

This call frees all resources allocated for this process by the Event Service on the event channel identified by the handle *channelHandle*. In particular, this call uninstalls any subscriptions of this process on the event channel and frees any resources allocated by the Event Service for the subscriptions. Additionally, this call frees events allocated for the process by *saEvtEventAllocate()* or *saEvtEventDeliverCallback()* (and that have not yet been freed by *saEvtEventFree()*) and frees memory allocated for the process by *saEvtEventAttributesGet()* that has not yet been freed by *saEvtEventPatternFree()*.

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.

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

The deletion (unlink) of an event channel (see Section 3.6.4) frees all resources allocated by the Event Service for it, such as published events with nonzero retention time.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saEvtChannelOpen(), *saEvtChannelOpenAsync()*, *SaEvtChannelOpenCallbackT*, *saEvtChannelUnlink()*, *saEvtEventSubscribe()*, *saEvtEventAllocate()*, *saEvtEventDeliverCallback()*, *saEvtEventAttributesGet()*, *saEvtEventFree()*, *saEvtEventPatternFree()*

3.6.4 saEvtChannelUnlink()

Prototype

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

Parameters

evtHandle - [*in*] The handle which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

channelName - [*in*] A pointer to the name of the event channel that is to be unlinked. The *SaNameT* type is defined in [1].

Description

This function deletes from the cluster an event channel designated by the name to which *channelName* points.

After successful completion of the invocation:

- The name to which *channelName* points is no longer valid, that is, any invocation of a function of the Event Service API that uses this event channel name returns an error unless an event channel is re-created with this name. The event channel is re-created by specifying in an *saEvtChannelOpen()* call or an *saEvtChannelOpenAsync()* call the SA_EVT_CHANNEL_CREATE flag and the same name of the event channel to be unlinked. 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 behavior 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 the only way to delete an existing event channel from the cluster namespace is by invoking *saEvtChannelUnlink()* on the event channel.

The deletion of an event channel frees all resources allocated by the Event Service for it, such as published events with nonzero 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 did not.

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

SA_AIS_ERR_BAD_HANDLE - The handle *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 the name to which *channelName* points does not exist.

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

See Also

saEvtInitialize(), *saEvtChannelClose()*

3.7 Event Operations

3.7.1 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 in an invocation of *saEvtChannelOpen()* or *saEvtChannelOpenCallback()*. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

eventHandle - [*out*] A pointer to the handle for the newly allocated event. It is the responsibility of the invoking process to allocate memory for this handle before invoking this function. The Event Service will assign the value of the handle when this function returns successfully. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

Description

The *saEvtEventAllocate()* function allocates memory for the event header and initializes all event attributes to default values, as described in Section 3.4.5.5 on page 21. The event allocated by *saEvtEventAllocate()* must be freed by invoking *saEvtEventFree()*.

This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_PUBLISHER flag was not set when the instance of the event channel identified by *channelHandle* was opened.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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).

SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_PUBLISHER flag was not set when the instance of the event channel identified by *channelHandle* was opened.

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

See Also

saEvtEventFree(), *saEvtEventPublish()*

3.7.2 saEvtEventFree()

Prototype

SaAisErrorT saEvtEventFree(

SaEvtEventHandleT eventHandle

);

Parameters

eventHandle - [*in*] The handle of the event whose memory can now be freed by the Event Service. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

Description

The function is used to free events allocated by *saEvtEventAllocate()* or obtained from *saEvtEventDeliverCallback()*.

The *saEvtEventFree()* function gives the Event Service permission to deallocate the event header and memory associated with the event attributes for the event identified by the handle *eventHandle*. If the event was allocated by the Event Service during a prior invocation of *saEvtEventDeliverCallback()*, *saEvtEventFree()* also gives the Event Service permission to deallocate the event data.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

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

3.7.3 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. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

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. The *SaEvtEventPatternArrayT* type is defined in Section 3.4.5.2 on page 20.

priority - [*in*] The priority of the event. The *SaEvtEventPriorityT* type is defined in Section 3.4.5.3 on page 21.

retentionTime - [*in*] The duration for which the event will be retained. The *SaTimeT* type is defined in [1].

publisherName - [*in*] A pointer to the name of the publisher of the event. The *SaNameT* type is defined in [1].

Description

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

Once the call to *saEvtEventAttributesSet()* returns, the memory for the structure to which *patternArray* points can be freed.

It is possible to invoke this API function on any event allocated by *saEvtEventAllocate()* or received from *saEvtEventDeliverCallback()* on the instance of the opened event channel on which *eventHandle* was obtained.

This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_PUBLISHER flag was not set when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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 when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

SA_AIS_ERR_TOO_BIG - One or more of the following conditions occur:

- The value of *patternArray->patternsNumber* is larger than the maximum value supported by the implementation. Refer to the description of the enum SA_EVT_MAX_NUM_PATTERNS_ID in Section 3.4.8 on page 27.
- The *patternSize* value of one or more patterns (*patternArray->patterns*[*i*].*patternSize* where *i* is an index to one of the patterns) is larger than the maximum value supported by the implementation. Refer to the description of the enum SA_EVT_MAX_PATTERN_SIZE_ID in Section 3.4.8 on page 27.
- The specified *retentionTime* exceeds the maximum retention time supported by the implementation. Refer to the description of the enum SA_EVT_MAX_RETENTION_DURATION_ID in Section 3.4.8 on page 27.

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

See Also

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

3.7.4 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. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

patternArray - [*in*/*out*] A pointer to a structure that describes the event pattern array and the number of patterns to be retrieved. The *SaEvtEventPatternArrayT* type is defined in Section 3.4.5.2 on page 20.

If the caller sets *patternArray->patterns* to NULL, the Event Service ignores the *patternArray->allocatedNumber* field and allocates memory for *patternArray->patterns* and each *patternArray->patterns*[*i*].*pattern* (where *i* is an index to one of the patterns). The Event Service sets the fields *patternArray->patternsNumber*, *patternArray->patterns*, *patternArray->patterns*[*i*].*pattern*, and *patternArray->patterns*[*i*].*patternSize* accordingly. The invoking process is then responsible for freeing the corresponding memory by specifying *patternArray->patterns* in a call to the *saEvtEventPatternFree()* function.

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* parame-

ters 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 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 occurs, it is unspecified whether some buffers to which *patternArray->patterns*[*i*].*pattern* point 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. The *SaEvtEventPriorityT* type is defined in Section 3.4.5.3 on page 21.

retentionTime - [*out*] A pointer to the duration for which the publisher will retain the event. The *SaTimeT* type is defined in [1].

publisherName - [*out*] A pointer to the name of the publisher of the event. The *SaNameT* type is defined in [1].

publishTime - [*out*] A pointer to the time at which the publisher published the event. The *SaTimeT* type is defined in [1].

eventId - [*out*] A pointer to the event identifier. The *SaEvtEventIdT* type is defined in Section 3.4.5.4 on page 21.

Description

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

If the NULL pointer is specified for any of the *out* or *in*/*out* parameters, the Event Service does not return the corresponding *out* value.

It is possible to invoke this API function on any event allocated by *saEvtEventAllocate()* or received from *saEvtEventDeliverCallback()* on the instance of the opened event channel on which *eventHandle* was obtained. This event may have been modified by the *saEvtEventAttributesSet()* function after the event has been allocated or received.

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 it allocated the event or the attributes set by a prior invocation of the *saEvtEventAttributesSet()* function.

This function fails with the SA_AIS_ERR_ACCESS error code if neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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_SPACE - The value of *allocatedNumber* in the structure to which *patternArray* points 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* pointer as an *in* parameter in the structure to which *patternArray* points is not NULL.

SA_AIS_ERR_ACCESS - Neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

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

See Also

saEvtEventPatternFree(), *SaEvtEventDeliverCallbackT*, *saEvtEventAllocate()*, *saEvtEventFree()*, *saEvtChannelOpen()*, *saEvtChannelOpenAsync()*, *saEvtEventAttributesSet()*, *saEvtFinalize()*

3.7.5 saEvtEventPatternFree()

Prototype

```
SaAisErrorT saEvtEventPatternFree(
        SaEvtEventHandleT eventHandle,
        SaEvtEventPatternT *patterns
);
```

Parameters

eventHandle - [*in*] The handle of the event whose attributes are to be retrieved. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

patterns - [*in*] A pointer to the memory that was allocated by the Event Service library in the *saEvtEventAttributesGet()* function and is to be deallocated. The *SaEvtEventPatternT* type is defined in Section 3.4.5.1 on page 20.

Description

This function frees the memory to which *patterns* points and which was allocated by the Event Service library in a previous call to the *saEvtEventAttributesGet()* function.

For details, refer to the description of the *patternArray* parameter of the *saEvtEventAttributesGet()* 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_BAD_HANDLE - The handle *eventHandle* is invalid, due to one or more of the reasons below:

- It is corrupted, was not obtained with 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_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

See Also

saEvtEventAttributesGet()

3.7.6 saEvtEventDataGet()

Prototype

```
SaAisErrorT saEvtEventDataGet(
        SaEvtEventHandleT eventHandle,
        void *eventData,
        SaSizeT *eventDataSize
);
```

Parameters

eventHandle - [*in*] The handle to the event that was delivered by *saEvtEventDeliverCallback()*. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

eventData - [*in*] A non-NULL pointer to a buffer (provided by the process) into which the Event Service stores the data associated with the delivered event.

eventDataSize - [*in*/*out*] The *in* value of the **eventDataSize* field is the size of the buffer to which *eventData* points, and which is provided by the invoking process. If this buffer is not large enough to hold all of the data associated with this event, no data will be copied into the buffer, and the value SA_AIS_ERR_NO_SPACE will be returned. The *out* value of the field to which *eventDataSize* points is valid only if the function returns either SA_AIS_OK or SA_AIS_ERR_NO_SPACE and is the size of the data associated with this event, which may be less than, equal to, or greater than the *in* value of the field to which *eventDataSize* points. The *SaSizeT* type is defined in [1].

Description

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

This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_SUBSCRIBER flag was not set when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which *eventHandle* was obtained.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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.

SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_SUBSCRIBER flag was not set when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

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

See Also

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

3.7.7 SaEvtEventDeliverCallbackT

Prototype

```
typedef void(*SaEvtEventDeliverCallbackT)(
        SaEvtSubscriptionIdT subscriptionId,
        SaEvtEventHandleT eventHandle,
        SaSizeT eventDataSize
);
```

Parameters

subscriptionId - [*in*] An identifier which a process supplied in an *saEvtEventSubscribe()* invocation, and which enables the process to determine which subscription resulted in the delivery of the event. The *SaEvtSubscriptionIdT* type is defined in Section 3.4.2 on page 18.

eventHandle - [*in*] The handle to the event delivered by this callback. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

eventDataSize - [*in*] The size of the data associated with the event. The *SaSizeT* type is defined in [1].

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 calling *saEvtDispatch()* on the handle *evtHandle* that was specified in the corresponding *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* call to obtain the channel handle to the instance of the event channel on which *eventHandle* was obtained.

A published event is received when it has an event pattern matching the filter of a subscription of this process. This filter is established by calling the *saEvtEventSubscribe()* function. For details on filtering, refer to Section 3.4.6 on page 22.

The process may invoke *saEvtEventAttributesGet()* to obtain the attributes associated with the event and *saEvtEventDataGet()* to obtain the data associated with the event.

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

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

To ensure that the *subscriptionId* provided to the event delivery callback is unique in applications that use the same event delivery callback function for subscriptions on more than one event channel, the application should provide *subscriptionIds* that are unique across all open event channels when calling the *saEvtEventSubscribe()* function.

Return Values

None

See Also

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

3.7.8 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 *saEvtEventAllocate()* or obtained from *saEvtEventDeliverCallback()*. If changes are required, the patterns must have been set by a prior call to *saEvtEventAttributesSet()*. The *SaEvtEventHandleT* type is defined in Section 3.4.1.2 on page 18.

eventData - [*in*] A pointer to a buffer that contains additional event information for the event being published. This parameter must be 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 to which *eventData* points. This parameter is ignored if *eventData* is NULL. The *SaSizeT* type is defined in [1].

eventId - [*out*] A pointer to an identifier of the event. The *SaEvtEventIdT* type is defined in Section 3.4.5.4 on page 21.

Description

If the *saEvtEventPublish()* function completes successfully, it publishes an event on the event channel on which the event identified by *eventHandle* was either allocated by invoking the *saEvtEventAllocate()* function or obtained from the *saEvtEventDeliverCallback()* function, and it returns the event identifier in the field to which *eventId* points. The event to be published consists of a standard set of attributes (the event header) and an optional data part.

Before an event is published, the publisher process may 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 by invoking *saEvtEventFree()* after *saEvtEventPublish()* returns.

This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_PUBLISHER flag was not set when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

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 did not.

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

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

- It is corrupted, was not obtained with 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 when either *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* was called to open the instance of the event channel on which the handle *eventHandle* was obtained.

SA_AIS_ERR_TOO_BIG - The total size of the event is larger than the maximum value supported by the implementation. Refer to the description of the enum SA_EVT_MAX_EVT_SIZE_ID in Section 3.4.8 on page 27.

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

See Also

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

3.7.9 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 in an invocation of *saEvtChannelOpen()* or *saEvtChannelOpenCallback()*. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

filters - [*in*] A pointer to an *SaEvtEventFilterArrayT* structure which defines filter patterns to be used to filter events received on the event channel. The process allocates this structure and may deallocate it and the memory for the filters when *saEvtEventSubscribe()* returns. The *SaEvtEventFilterArrayT* type is defined in Section 3.4.6.3 on page 23.

subscriptionId - [*in*] An identifier that uniquely identifies a particular subscription on the instance of the opened event channel designated by *channelHandle*. This identifier is used as a parameter of the *saEvtEventDeliverCallback()* function. The *SaEvtSubscriptionIdT* type is defined in Section 3.4.2 on page 18.

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 by the invocation of the *saEvtEventDeliverCallback()* callback function, which must have been supplied when the process called the *saEvtInitialize()* function.

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

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

This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_SUBSCRIBER flag was not set when the instance of the event channel identified by *channelHandle* was opened.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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 invocation of *saEvtInitialize()* to initialize the Event Service was incomplete, since the *saEvtEventDeliverCallback()* callback function is missing.

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_EXIST - A subscription using the same *subscriptionId* already exists on the instance of the opened event channel designated by *channelHandle*.

SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_SUBSCRIBER flag was not set when the instance of the event channel identified by *channelHandle* was opened.

SA_AIS_ERR_TOO_BIG - The value of *filtersNumber* in the structure to which the *filters* parameter points or the length of one or more filter strings exceeds the maximum size supported by the implementation. Refer to the description of the enums SA_EVT_MAX_NUM_PATTERNS_ID and SA_EVT_MAX_PATTERN_SIZE_ID in Section 3.4.8 on page 27.

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

See Also

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

3.7.10 saEvtEventUnsubscribe()

Prototype

SaAisErrorT saEvtEventUnsubscribe(

SaEvtChannelHandleT channelHandle,

SaEvtSubscriptionIdT subscriptionId

);

Parameters

channelHandle - [*in*] The handle of the event channel on which the subscriber is requesting the Event Service to delete the subscription. The *channelHandle* parameter must have been obtained previously in an invocation of *saEvtChannelOpen()* or *saEvtChannelOpenCallback()*. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

subscriptionId - [*in*] An identifier that uniquely identifies a particular subscription on the instance of the opened event channel designated by *channelHandle*. The *SaEvtSubscriptionIdT* type is defined in Section 3.4.2 on page 18.

Description

The *saEvtEventUnsubscribe()* function allows a process to stop receiving events for a particular subscription on an event channel by removing the subscription. The subscription is identified by the pair *channelHandle* and *subscriptionId* and must have been set up by a previous invocation of the *saEvtEventSubscribe()* function for the same pair.

It is an error if the value of the *subscriptionId* parameter does not match a previously registered subscription on the instance of the opened event channel identified by *channelHandle*.

The *saEvtEventUnsubscribe()* function purges events that are queued to be delivered to the process and that no longer match any subscription because of the removal of the subscription identified by the pair *channelHandle* and *subscriptionId*.

A process that wants 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 did not.

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

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

- It is corrupted, was not obtained with 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 *subscriptionId* parameter does not match any currently registered subscription on the instance of the opened event channel designated by *channelHandle*.

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

See Also

saEvtEventSubscribe()

3.7.11 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 *channelHandle* must have been obtained previously in an invocation of *saEvtChannelOpen()* or *saEvtChannelOpenCallback()*. The *SaEvtChannelHandleT* type is defined in Section 3.4.1.3 on page 18.

eventId - [*in*] The identifier of the event. The *SaEvtEventIdT* type is defined in Section 3.4.5.4 on page 21.

Description

The *saEvtEventRetentionTimeClear()* function is used to clear the retention time of the 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.

This function fails with the SA_AIS_ERR_ACCESS error code if neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when one of the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions was called to open the instance of the event channel identified by *channelHandle*.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained with 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.

SA_AIS_ERR_ACCESS - Neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when one of the *saEvtChannelOpen()* or *saEvtChannelOpenAsync()* functions was called to open the instance of the event channel identified by *channelHandle*.

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

See Also

saEvtEventPublish(), *SaEvtEventDeliverCallbackT*

3.8 Limit Fetch API

3.8.1 saEvtLimitGet()

Prototype

```
SaAisErrorT saEvtLimitGet(
        SaEvtHandleT evtHandle,
        SaEvtLimitIdT limitId,
        SaLimitValueT *limitValue
);
```

Parameters

evtHandle - [*in*] The handle which was obtained by a previous invocation of the *saEvtInitialize()* function and which designates this particular initialization of the Event Service. The *SaEvtHandleT* type is defined in Section 3.4.1.1 on page 18.

limitId - [*in*] The Event Service limit whose implementation-specific value is to be queried. The limits are defined in the *SaEvtLimitIdT* type in Section 3.4.8 on page 27.

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

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. This error is returned due to one or both of the following reasons:

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

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

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

See Also

saEvtInitialize()

4 Event Service UML Information Model

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

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

The Event Service UML class diagram has one class, which shows the contained attributes.

4.1 DN Format for the Event Service UML Class

Table 3 DN Formats for Objects of the Event Service Class

Object Class	DN Format for Objects of the Class
SaEvtChannel	"*safChnl*=…,* "

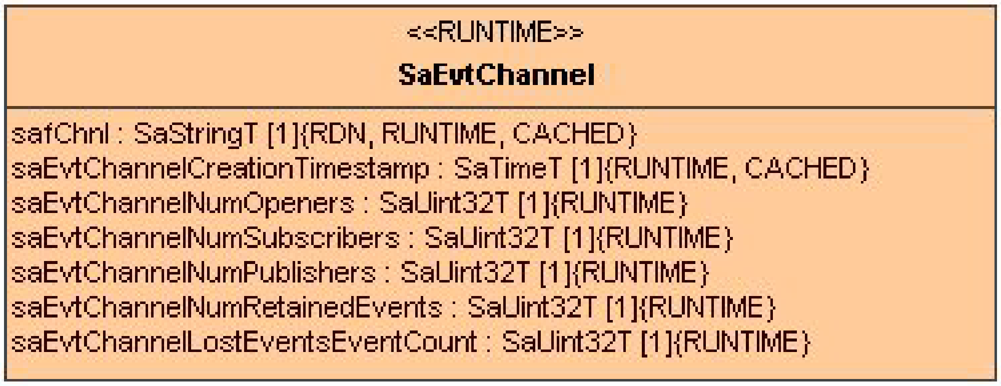
The ‘*’ notation at the end of a DN format indicates that zero, one or more RDNs may be appended to the DN format.

4.2 Event Service UML Class

The Event Service UML class diagram contains one class, *SaEvtChannel*, which is a runtime object class exposing various runtime attributes of an event channel.

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

FIGURE 1 Event Service UML Class


<<RUNTIME>>
SaEvtChannel
safChnl : SaStringT [1]{RDN, RUNTIME, CACHED}
saEvtChannelCreationTimestamp : SaTimeT [1]{RUNTIME, CACHED}
saEvtChannelNumOpeners : SaUint32T [1]{RUNTIME}
saEvtChannelNumSubscribers : SaUint32T [1]{RUNTIME}
saEvtChannelNumPublishers : SaUint32T [1]{RUNTIME}
saEvtChannelNumRetainedEvents : SaUint32T [1]{RUNTIME}
saEvtChannelLostEventsEventCount : SaUint32T [1]{RUNTIME}

5 Event Service Administration API

The Event Service has no administration interface at the time of publication of this specification.

6 Alarms and Notifications

The Event Service does not issue any alarms and notifications at the time of publication of this specification.

7 Event Service Management Interface

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

7.1 Event Service MIB (SAF-EVT-SVC-MIB)

The Event Service MIB contains the single read-only table *saEvtChannelTable*, which contains attributes for the currently created event channels in the cluster, which include all event channels in the cluster that have not been unlinked as well as the ones that have been unlinked but are still in-use within the cluster.

This table mimics the UML runtime object class *SaEvtChannel*, as described in Section 4.2 in terms of the objects contained in the table.

Additionally, the Event Service MIB also defines SNMP traps that correspond to the various notifications for the service, which are described in Chapter 6 of this specification.

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

Index of Definitions