Service Availability[™] Forum Application Interface Specification

Event Service

SAI-AIS-EVT-B.03.01



This specification was reissued on **September 30, 2011** under the Artistic License 2.0. The technical contents and the version remain the same as in the original specification.

•



SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT	1
The Service Availability [™] Forum Application Interface Specification (the "Package") found at the URL http://www.saforum.org is generally made available by the Service Availability Forum (the "Copyright Holder") for use in developing products that are compatible with the standards provided in the Specification. The terms and conditions which govern the use of the Package are covered by the Artistic License 2.0 of the Perl Foundation, which is reproduced here.	
The Artistic License 2.0	5
Copyright (c) 2000-2006, The Perl Foundation.	
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.	
Preamble	
This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed.	10
The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.	10
You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.	
Definitions	15
"Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.	15
"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.	
"You" and "your" means any person who would like to copy, distribute, or modify the Package.	
"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.	20
"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.	
"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.	
"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.	25
"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.	20
"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.	
"Source" form means the source code, documentation source, and configuration files for the Package.	
"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.	30
Permission for Use and Modification Without Distribution	
(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.	
Permissions for Redistribution of the Standard Version	
(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.	35
(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.	
Distribution of Modified Versions of the Package as Source	
(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:	40

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

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

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

(i) the Original License or

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

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

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

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

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

Aggregating or Linking the Package

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

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

Items That are Not Considered Part of a Modified Version

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

General Provisions

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

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

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

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

(14) Disclaimer of Warranty:

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

35

40



20

1

5

10

15

25



Table of Contents

Event Service

1 Document Introduction	
1.1 Document Purpose	5
1.2 AIS Documents Organization	Ũ
1.3 History	
1.3.1 New Topics	
1.3.2 Clarifications	
1.3.3 Changes in Return Values of API Functions:	10
1.3.4 Removed Topics	10
1.3.5 Other Changes	
1.4 References	
1.5 How to Provide Feedback on the Specification	
1.6 How to Join the Service Availability TM Forum	4 -
1.7 Additional Information	15
1.7.1 Member Companies	
1.7.2 Press Materials	
2 Overview	20
2.1 Event Service	20
3 SA Event Service API	
3.1 Event Service Model	
3.1.1 Events	25
3.1.2 Event Channels	
3.1.3 Event Filtering	
3.2 Unavailability of the Event Service API on a Non-Member Node	
3.2.1 A Member Node Leaves or Rejoins the Cluster Membership	
3.2.2 Guidelines for Event Service Implementers	30
3.3 Include File and Library Name	00
3.4 Type Definitions	
3.4.1 Handles	
3.4.1.1 SaEvtHandleT	
3.4.1.2 SaEvtEventHandleT	25
3.4.1.3 SaEvtChannelHandleT	35
3.4.2 SaEvtSubscriptionIdT	
3.4.3 SaEvtCallbacksT	
3.4.4 SaEvtChannelOpenFlagsT 19 3.4.5 Event Patterns and Attributes 19	
3.4.5 Event Patterns and Attributes	
3.4.5.1 SaEvtEventPatternArrayT	40
3.4.5.3 SaEvtEventPriorityT	
3.4.5.4 SaEvtEventIdT	



3.4.5.5 Event Attributes	1
3.4.6 Event Filters	
3.4.6.1 SaEvtEventFilterTypeT 23	
3.4.6.2 SaEvtEventFilterT	
3.4.6.3 SaEvtEventFilterArrayT	5
3.4.7 "Lost Event" Event	•
3.4.8 SaEvtLimitIdT	
3.5 Library Life Cycle	
3.5.1 saEvtInitialize()	
3.5.2 saEvtSelectionObjectGet()	10
3.5.3 saEvtDispatch()	10
3.5.4 saEvtFinalize()	
3.6 Event Channel Operations	
3.6.1 saEvtChannelOpen() and saEvtChannelOpenAsync()	
3.6.2 SaEvtChannelOpenCallbackT	4 -
3.6.3 saEvtChannelClose()403.6.4 saEvtChannelUnlink()42	15
\checkmark	
3.7 Event Operations	
3.7.1 saEvtEventAllocate()	
3.7.2 saEvtEventFree() 46 3.7.3 saEvtEventAttributesSet() 47	
3.7.4 saEvtEventAttributesGet()	20
3.7.5 saEvtEventPatternFree()	
3.7.6 saEvtEventDataGet()	
3.7.7 SaEvtEventDeliverCallbackT	
3.7.8 saEvtEventPublish()	
3.7.9 saEvtEventSubscribe()	25
3.7.10 saEvtEventUnsubscribe()	
3.7.11 saEvtEventRetentionTimeClear()	
3.8 Limit Fetch API	
3.8.1 saEvtLimitGet()	
5.0.1 Sull vielmittoet()	30
4 Event Service UML Information Model	
4.1 DN Format for the Formit Commiss UNIL Class	
4.1 DN Format for the Event Service UML Class	
4.2 Event Service UML Class	
5 Event Service Administration API	35
6 Alarms and Notifications	
7 Event Service Management Interface	
7.1 Event Service MIB (SAF-EVT-SVC-MIB)	40
Index of Definitions	



1	Document Introduction	1
1.1	Document Purpose	
	This document defines the Event Service of the Application Interface Specification (AIS) of the Service Availability [™] 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.	5 10
	Typically, the Service Availability [™] Forum Application Interface Specification will be used in conjunction with the Service Availability [™] Forum Hardware Interface Specification (HPI).	15
1.2	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]).	20
1.3	B History	
	Previous releases of the Event Service specification:	
	(1) SAI-AIS-EVT-A.01.01	25
	(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.	30
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]). 	35
	 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 Ser- vice. 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. 	40



 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. 	1
 Chapter 7 presents the Event Service Management Interface. 	5
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. 	10
 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. 	15
 The description of the saEvtChannelClose() function (see Section 3.6.3 on page 40) clarifies which resources this function frees for the invoking process. 	15
 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	25
All API functions except saEvtFinalize() and SaEvtEventDeliverCallbackT	SA_AIS_ERR_UNAVAILABLE	new	
saEvtChannelOpen(), saEvtChannelOpenAsync(), and SaEvtChannelOpenCallbackT	SA_AIS_ERR_NO_RESOURCES	clarified	30
saEvtEventAttributesSet()	SA_AIS_ERR_TOO_BIG	extended	
saEvtEventPublish() and saEvtEventSubscribe()	SA_AIS_ERR_TOO_BIG	clarified	35
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

40



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	1
Service B.03.01 version.	5
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 Ser- vice B.03.01 version.	10
1.3.5 Other Changes	
In Section 3.4.5.4 on page 21 on the <i>SaEvtEventIdT</i> type, "LL" has been appended to the numeric values of constants.	15
1.4 References	
The following documents contain information that is relevant to the specification:	20
 [1] Service Availability[™] Forum, Service Availability Interface, Overview, SAI-Overview-B.03.01 	
[2] Service Availability [™] Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.02.01	05
[3] Service Availability [™] Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.02.01	25
[4] Service Availability [™] Forum, Application Interface Specification, Cluster Mem- bership Service, SAI-AIS-CLM-B.03.01	
[5] Service Availability [™] Forum, SA Forum Information Model in XML Metadata Interchange (XMI) v2.1 format, SAI-XMI-A.02.01	30
[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.	35
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[™] 40 Forum website (<u>http://www.saforum.org</u>).



You can also sign up to receive information updates on the Forum or the Specifica- tion.	1
1.6 How to Join the Service Availability™ Forum	
The Promoter Members of the Forum require that all organizations wishing to partici- pate in the Forum complete a membership application. Once completed, a represen- tative of the Service Availability [™] Forum will contact you to discuss your membership in the Forum. The Service Availability [™] Forum Membership Application can be com- pleted online by following the pertinent links provided on the Forum's website (<u>http://www.saforum.org</u>).	5 10
You can also submit information requests online. Information requests are generally responded to within three business days.	15
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 (<u>http://www.saforum.org</u>).	20
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 (<u>http://www.saforum.org</u>).	25
	30



2	Overview
4	

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 10 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 15 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).

20

1

5

25



5
10
15
20
25
30
35



3 SA Event Service API	1
3.1 Event Service Model	
3.1.1 Events	5
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 <i>saEvtEventAllocate()</i> function to allocate the event header and the <i>saEvtEventFree()</i> function to deallocate it. The <i>saEvtEventAllocate()</i> function returns a handle that can be used in subsequent invocations of functions provided by the Event Service API.	10
A process writes the event attributes by invoking the <i>saEvtEventAttributesSet()</i> func- tion and reads these attributes by invoking the <i>saEvtEventAttributesGet()</i> function.	15
An event is published by invoking the <i>saEvtEventPublish()</i> 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 <i>saEvtEventPublish()</i> 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.	20 25
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 <i>saEvtEventSubscribe()</i> function to subscribe for events on this event channel. A process opens an event channel by	30
subscribe for events on this event channel. A process opens an event channel by invoking one of the <i>saEvtEventChannelOpen()</i> or <i>saEvtEventChannelOpenAsync()</i> 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.	35

A process can open an event channel to publish events and to subscribe to receive 40 events. Publishers can also be subscribers on the same event channel. Event channels can be deleted by invoking the *saEvtChannelUnlink()* function.

SERVICE VAILABILITY

Once an event has been allocated for an event channel by invoking the 1 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: 5 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 10 quaranteed 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 node¹ 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 15 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 20 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 25 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 30 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. 35 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]).



5

10

15

20

40

• **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: 25 "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.	1	
	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.	5	
	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.	10	
	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. 	15	
	In this case, the process will receive all notification events related to the specified table in the specified database for any primary key.	20	
3.2 Unavailability of the Event Service API on a Non-Member Node			
	The Event Service <u>does not</u> provide service to processes on cluster nodes that are not in the cluster membership (see [4]).	25	
	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.	30	
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 administra- tively 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:	35	
	⇒ Calls to <i>saEvtInitialize()</i> will fail with SA_AIS_ERR_UNAVAILABLE.		
	⇒ All Event Service APIs that are invoked by the process and that operate on han- dles already acquired by the process will fail with SA_AIS_ERR_UNAVAILABLE with the following exceptions, assuming that the handle <i>evtHandle</i> has already been acquired:	40	



	 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; oth- erwise, the callback will not be called. 	1 5
	 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.	10
	\Rightarrow The callback <i>SaEvtEventDeliverCallbackT</i> 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 <i>saEvtFinalize()</i> , which is used to free the library handles and all resources associated with these handles. Hence, it is recom-	15
	•	20
	When the node leaves the membership, the Event Service executing on the remain- ing nodes of the cluster behaves as if all processes that were using the Event Service	25
3.2.2 0	Buidelines for Event Service Implementers	
	bership 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	30 35
	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 40 must be included in the source of an application using the Event Service API:



#include <saevt.h></saevt.h>	1
To use the Event Service API, an application must be bound with the following library:	
libSaEvt.so	5
3.4 Type Definitions	
The Event Service uses the types described in the following sections.	
3.4.1 Handles	10
3.4.1.1 SaEvtHandleT	
typedef SaUint64T SaEvtHandleT;	15
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.	10
3.4.1.2 SaEvtEventHandleT	20
typedef SaUint64T SaEvtEventHandleT;	
This type is used for the handle to an event.	
3.4.1.3 SaEvtChannelHandleT	25
typedef SaUint64T SaEvtChannelHandleT;	
This type is used for the handle to an open event channel.	
3.4.2 SaEvtSubscriptionIdT	30
typedef SaUint32T SaEvtSubscriptionIdT;	
This type is used for an identifier representing a particular subscription by a particu- lar 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.	35
3.4.3 SaEvtCallbacksT	
The SaEvtCallbacksT type is defined as follows:	40



	typedef struct {	1	
	SaEvtChannelOpenCallbackT saEvtChannelOpenCallback;		
	SaEvtEventDeliverCallbackT saEvtEventDeliverCallback;		
}	} SaEvtCallbacksT;	5	
	A structure of the <i>SaEvtCallbacksT</i> type (called a callbacks structure) is used to specify the callback functions that the Event Service can invoke.		
3.4.4 S	aEvtChannelOpenFlagsT	10	
#	#define SA_EVT_CHANNEL_PUBLISHER 0X1		
#	#define SA_EVT_CHANNEL_SUBSCRIBER 0X2		
#	#define SA_EVT_CHANNEL_CREATE 0X4	15	
ť	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. 		
S	When an event channel is opened by invoking either <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> , some combination of these flags are bitwise ORed together to provide the value of the <i>channelOpenFlags</i> .		
3.4.5 E	event Patterns and Attributes	30	
r c	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).	25	
		35	



3.4.5.1 SaEvtEventPatternT	1
typedef struct {	
SaSizeT allocatedSize;	5
SaSizeT patternSize;	0
SaUint8T *pattern;	
} SaEvtEventPatternT;	
In the context of the <i>saEvtEventAttributesGet()</i> function, these fields are used as fol- lows:	10
 allocatedSize (in): size of the buffer allocated to receive the pattern value 	
 patternSize (out): actual size of the pattern of the received event 	. –
• <i>pattern</i> (<i>out</i>): pointer to a buffer to which the pattern value will be copied	15
In the context of the <i>saEvtEventAttributesSet()</i> or <i>saEvtEventSubscribe()</i> functions, these fields are used as follows:	20
allocatedSize: ignored	20
 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	25
typedef struct {	
SaSizeT allocatedNumber;	
SaSizeT patternsNumber;	30
SaEvtEventPatternT *patterns;	
} SaEvtEventPatternArrayT;	
In the context of the <i>saEvtEventAttributesGet()</i> function, these fields are used as fol- lows:	35
allocatedNumber (in): number of entries allocated in the patterns buffer	
 patternsNumber (out): actual number of patterns in the event 	40
 patterns (out): pointer to a buffer to which the array of patterns will be copied 	



In the context of the <i>saEvtEventAttributesSet()</i> function, these fields are used as lows:	s fol- 1
 <i>allocatedNumber</i>: ignored <i>patternsNumber</i> (<i>in</i>): number of patterns in the patterns array <i>patterns</i> (<i>in</i>): pointer to the array of patterns 	5
3.4.5.3 SaEvtEventPriorityT	
#define SA_EVT_HIGHEST_PRIORITY 0 #define SA_EVT_LOWEST_PRIORITY 3	10
typedef SaUint8T SaEvtEventPriorityT;	
3.4.5.4 SaEvtEventIdT	15
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	20
Event identifier for an allocated but not yet published event.	
#define SA_EVT_EVENTID_LOST 1LL	
Event identifier for a "lost event".	25
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 event handle returned by these functions in a call to saEvtEventAttributesGet().	g the 30
A process may not access the event attributes of a published event (which is ow by the Event Service), except when discarding a published event with nonzero re tion time by invoking the <i>saEvtEventRetentionTimeClear()</i> function to clear the re tion time of the event. For this purpose, the process uses the event id that it obta in a previous <i>saEvtEventPublish()</i> call to refer to the published event.	eten- eten-
The following list shows all event attributes. For each attribute, it is specified when the process in which the event resides has write access or not, as some attribute can be set only by the Event Service. Additionally, the list gives the default value each attribute of an event allocated by invoking the <i>saEvtEventAllocate()</i> function	es e of 40



 Event Pattern Array - An array defined earlier by the SaEvtEventPatternArrayT structure. Write access is permitted. Default: no patterns 	1
 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 	5
 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 	10
 Event Retention Time - The retention time is the duration for which the event is retained. Write access is permitted. Default: 0 	15
• 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	20
(including zeros) may be used as the name. Write access is permitted. Default: empty string (<i>SaNameT.length</i> = 0)	25
 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. 	30
This attribute is read-only. Default: SA_EVT_EVENTID_NONE	
3.4.6 Event Filters	35
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	1	
typedef enum {		
$SA_EVT_PREFIX_FILTER = 1,$	5	
$SA_EVT_SUFFIX_FILTER = 2,$	Ũ	
$SA_EVT_EXACT_FILTER = 3,$		
SA_EVT_PASS_ALL_FILTER = 4		
} SaEvtEventFilterTypeT;	10	
The <i>saEvtEventFilterTypeT</i> 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	15	
typedef struct {		
SaEvtEventFilterTypeT filterType;		
SaEvtEventPatternT filter;	20	
} 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	25	
typedef struct {		
SaSizeT filtersNumber;	30	
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	40	
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 <i>saEvtEventSubscribe()</i> call.		



Table 2 Filter Types and Pattern Matching Algorithms		
Filter Type	Matching Algorithm	
SA_EVT_PREFIX_FILTER	The entire filter must match the first <i>filter.patternSize</i> charac- ters 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="abcdxyz" Non-Match example: Filter="xyz", Event Pattern="xyzab" Non-Match example: Filter="xyz", Event Pattern="xyzab"	5
SA_EVT_SUFFIX_FILTER	The entire filter must match the last <i>filter.patternSize</i> charac- ters 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 pat- tern; only the last two characters match.)	15 20
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.)	25
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.	30

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".

35



If the patterns of an event match the filters of several different subscriptions of a par- ticular subscriber on a single event channel, the Event Service invokes <i>saEvtEventDeliverCallback()</i> (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 chan- nels, the Event Service invokes <i>saEvtEventDeliverCallback()</i> twice (one for each opened channel).	1 5
3.4.7 "Lost Event" Event	
A subscriber can lose events in any of the following situations:	10
 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.	15
 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" noti- fies 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 sub- scriber, regardless of the filters that the subscriber has set.	20
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.	25
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.	30
The Event Service sets the attributes of the "lost event" event as follows:	00
 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"	35
 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. 	40
 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. 	1
	5
	10
	15
	20
	25
	30
	35
	40



SaEvtLimitIdT	
The SaEvtLimitIdT enum provides a set of valu implementation of the Event Service. Note that not define a configuration for these limits, which mentation.	the Event Service specification does
The user can retrieve at runtime the current val the identifier of the limit (one of the enum value below) when invoking the <i>saEvtLimitGet()</i> funct	s of the type SaEvtLimitIdT, defined
The limit value is returned in a parameter of a g defined in [1]). As the limit identified by the enu SA_EVT_MAX_RETENTION_DURATION_ID is of <i>SaLimitValueT</i> must be used for further acce	m s of type <i>SaTimeT</i> , the <i>timeValue</i> field
specification are of type <i>SaUint64T</i> , the <i>uint64</i> 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_DURATIO	$DN_ID = 5,$
} SaEvtLimitIdT;	
The values of the SaEvtLimitIdT enumeration ty	ype have the following interpretation:
 SA_EVT_MAX_NUM_CHANNELS_ID - Th maximum number of event channels in the 	
 SA_EVT_MAX_EVT_SIZE_ID - This enum size in bytes of the event. This size include of the data. 	
 SA_EVT_MAX_PATTERN_SIZE_ID - This maximum size in bytes of a pattern of an e 	
 SA_EVT_MAX_NUM_PATTERNS_ID - This maximum number of patterns an event car ber of filters that can be specified is the sa terns an event can have. 	have. Note that the maximum num-
 SA_EVT_MAX_RETENTION_DURATION retrieve the longest period an event will be 	



3.5 Library Life Cycle	1	
3.5.1 saEvtInitialize()		
Prototype	5	
SaAisErrorT saEvtInitialize(
SaEvtHandleT *evtHandle,		
const SaEvtCallbacksT *evtCallbacks,	10	
SaVersionT *version		
);		
Parameters	15	
<i>evtHandle</i> - [<i>out</i>] 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 <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.		
<i>evtCallbacks</i> - [<i>in</i>] If <i>evtCallbacks</i> is set to NULL, no callback is registered; If <i>evtCallbacks</i> is not set to NULL, it is a pointer to an <i>SaEvtCallbacksT</i> 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 <i>SaEvtCallbacksT</i> type is defined in Section 3.4.3 on page 18.	20 25	
<i>version</i> - [<i>in/out</i>] As an input parameter, <i>version</i> is a pointer to a structure containing the required Event Service version. In this case, <i>minorVersion</i> is ignored and should be set to 0x00. As an output parameter, <i>version</i> is a pointer to a structure containing the version actually supported by the Event Service. The <i>SaVersionT</i> type is defined in [1].	30	
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 <i>evtHandle</i> 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.	35	
If the implementation supports the specified <i>releaseCode</i> and <i>majorVersion</i> , SA_AIS_OK is returned. In this case, the structure pointed to by the <i>version</i> parame- ter is set by this function to:	40	



 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 <i>version</i> parameter is set to:
if (implementation supports the required releaseCode)
releaseCode = required releaseCode
else {
if (implementation supports <i>releaseCode</i> higher than the required <i>releaseCode</i>)
<i>releaseCode</i> = the lowest value of the supported release codes that is higher than the required <i>releaseCode</i>
else
<i>releaseCode</i> = the highest value of the supported release codes that is lower than the required <i>releaseCode</i>
}
<i>majorVersion</i> = highest value of the major versions that this implementation can support for the returned <i>releaseCode</i>
<i>minorVersion</i> = highest value of the minor versions that this implementation can support for the returned values of <i>releaseCode</i> and <i>majorVersion</i>
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.	1
	SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).	5
	SA_AIS_ERR_VERSION - The version provided in the structure to which the <i>version</i> 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.	10
	See Also	
	saEvtSelectionObjectGet(), saEvtDispatch(), saEvtFinalize()	4 5
3.5.2	saEvtSelectionObjectGet()	15
	Prototype	
	SaAisErrorT saEvtSelectionObjectGet(20
	SaEvtHandleT evtHandle,	
	SaSelectionObjectT *selectionObject	
);	25
	Parameters	_0
	<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Event Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.	30
	<i>selectionObject</i> - [<i>out</i>] A pointer to the operating system handle that the invoking pro- cess can use to detect pending callbacks. The <i>SaSelectionObjectT</i> type is defined in [1].	25
	Description	35
	This function returns the operating system handle associated with the handle <i>evtHandle</i> . The invoking process can use this operating system handle to detect pending callbacks, instead of repeatedly invoking the <i>saEvtDispatch()</i> function for this purpose.	40



In a POSIX environment, the operating system handle is a file descriptor that is used with the <i>poll()</i> or <i>select()</i> system calls to detect incoming callbacks.	1
The operating system handle returned by <i>saEvtSelectionObjectGet()</i> is valid until <i>saEvtFinalize()</i> is invoked on the same handle <i>evtHandle</i> .	5
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.	10
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 pro- cess may retry later.	15
SA_AIS_ERR_BAD_HANDLE - The handle <i>evtHandle</i> is invalid, since it is corrupted, uninitialized, or has already been finalized.	
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	20
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).	25
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	30
saEvtInitialize(), saEvtDispatch(), saEvtFinalize()	

SERVICE AVAILABILITY[®] FORUM

3.5.3 saEvtDispatch()		1
	Prototype	
	SaAisErrorT saEvtDispatch(5
	SaEvtHandleT evtHandle,	
	SaDispatchFlagsT dispatchFlags	
);	10
	Parameters	10
	<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Event Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.	15
	<i>dispatchFlags</i> - [<i>in</i>] Flags that specify the callback execution behavior of the <i>saEvtDispatch()</i> function, which have the values SA_DISPATCH_ONE, SA_DISPATCH_ALL, or SA_DISPATCH_BLOCKING. These flags are values of the <i>SaDispatchFlagsT</i> enumeration type, which is described in [1].	20
	Description	
	In the context of the calling thread, this function invokes pending callbacks for the handle <i>evtHandle</i> in the way specified by the <i>dispatchFlags</i> parameter.	25
	Return Values	
	SA_AIS_OK - The function completed successfully. This value is also returned if this function is being invoked with <i>dispatchFlags</i> set to SA_DISPATCH_ALL or SA_DISPATCH_BLOCKING, and the handle <i>evtHandle</i> has been finalized.	30
	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.	35
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
	SA_AIS_ERR_BAD_HANDLE - The handle <i>evtHandle</i> is invalid, since it is corrupted, uninitialized, or has already been finalized.	40
	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.	1
	See Also	
	saEvtInitialize(), saEvtFinalize()	5
3.5.4 s	saEvtFinalize()	
	Prototype	10
	SaAisErrorT saEvtFinalize(
	SaEvtHandleT evtHandle	
);	15
	Parameters	10
	<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Event Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.	20
	Description	
	The <i>saEvtFinalize()</i> function closes the association represented by the <i>evtHandle</i> parameter between the invoking process and the Event Service. The process must have invoked <i>saEvtInitialize()</i> before it invokes this function. A process must invoke this function once for each handle it acquired by invoking <i>saEvtInitialize()</i> .	25
	If the <i>saEvtFinalize()</i> function completes successfully, it releases all resources acquired when the <i>saEvtInitialize()</i> function was called. Moreover, it closes all event channels that are open for the particular handle. Furthermore, it cancels all pending <i>SaEvtChannelOpenCallbackT</i> 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.	30
	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 para- graph.	35
	After <i>saEvtFinalize()</i> completes successfully, the handle <i>evtHandle</i> and the selection object associated with it are no longer valid.	
	Return Values	40

SA_AIS_OK - The function completed successfully.

SERVICE AVAILABILITY FORUM

SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	1
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.	5
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle evtHandle is invalid, since it is corrupted, uninitialized, or has already been finalized.	10
See Also	
saEvtInitialize(), saEvtChannelClose(), SaEvtChannelOpenCallbackT, saEvtSelectionObjectGet()	15

20

25

30

35



3.6 Event Channel Operations		
3.6.1 saEvtChannelOpen() and saEvtChannelOpenAsync()		
Prototype		5
SaAisErrorT saEvt	ChannelOpen(
SaEvtHandle	eT evtHandle,	
const SaNar	neT *channelName,	10
SaEvtChann	elOpenFlagsT channelOpenFlags,	
SaTimeT tim	neout,	
SaEvtChann	elHandleT *channelHandle	4 5
);		15
SaAisErrorT saEvt	ChannelOpenAsync(
SaEvtHandle	eT evtHandle,	
Salnvocatio	nT invocation,	20
const SaNar	neT *channelName,	
SaEvtChanr	elOpenFlagsT channelOpenFlags	
);		25
Parameters		
<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Event Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.		30
<i>invocation</i> - [<i>in</i>] This parameter allows the invoking process to match this invocation of <i>saEvtChannelOpenAsync()</i> with the corresponding callback call. The <i>SaInvocationT</i> type is defined in [1].		35
<i>channelName</i> - [<i>in</i>] A pointer to the name of the event channel that identifies an event channel globally in a cluster. The <i>SaNameT</i> type is defined in [1].		
value of this param SA_EVT_CHANNE	- [<i>in</i>] The requested access modes of the event channel. The eter is obtained by a bitwise OR of the L_PUBLISHER, SA_EVT_CHANNEL_SUBSCRIBER, and L_CREATE flags, as defined for the <i>SaEvtChannelOpenFlagsT</i>	40

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 5 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 30 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 35 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.

20

1

10

15

25



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 <i>timeout</i> 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 pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>evtHandle</i> is invalid, since it is corrupted, uninitialized, or has already been finalized.	
SA_AIS_ERR_INIT - The previous invocation of <i>saEvtInitialize()</i> to initialize the Event Service was incomplete, since the <i>saEvtChannelOpenCallback()</i> callback function is missing. This applies only to the <i>saEvtChannelOpenAsync()</i> 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 <i>channelOpenFlags</i> , and the name to which <i>channelName</i> points is not a DN, or the type of its first RDN is not <i>safChnl</i> .	
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 <i>channelName</i> 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)(5	
SaInvocationT invocation,		
SaEvtChannelHandleT channelHandle,		
SaAisErrorT error	10	
);	10	
Parameters		
<i>invocation</i> - [<i>in</i>] This parameter was supplied by a process in the corresponding invo- cation of the <i>saEvtChannelOpenAsync()</i> function and is used by the Event Service in this callback. The <i>invocation</i> parameter allows the process to match the invocation of that function with this callback. The <i>SaInvocationT</i> type is defined in [1].	15	
<i>channelHandle</i> - [<i>in</i>] The handle that designates the event channel. The <i>SaEvtChannelHandleT</i> type is defined in Section 3.4.1.3 on page 18.	20	
<i>error</i> - [<i>in</i>] This parameter indicates whether the <i>saEvtChannelOpenAsync()</i> function was successful. The <i>SaAisErrorT</i> type is defined in [1]. The values that can be returned are:	25	
 SA AIS OK - The function completed successfully. 	20	
 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. 	30	
 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 cor- rupted, uninitialized, or has already been finalized. 	35	
 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. 	40	



 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. 	1
 SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory). In particular, this value is returned if the number of event chan- nels 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. 	5
 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. 	10
 SA_AIS_ERR_BAD_FLAGS - The channelOpenFlags parameter in the corre- sponding invocation of the saEvtChannelOpenAsync() function is invalid. 	
 SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavail- able on this cluster node because it is not a member node. 	15
Description	
The Event Service invokes this callback function when the operation requested by the invocation of <i>saEvtChannelOpenAsync()</i> completes.	20
This callback is invoked in the context of a thread calling <i>saEvtDispatch()</i> on the han- dle <i>evtHandle</i> that was specified in the <i>saEvtChannelOpenAsync()</i> call.	
If this function succeeds, the handle to the opened/created event channel is returned in <i>channelHandle</i> ; otherwise, an error is returned in the <i>error</i> parameter.	25
Return Values	
None	20
See Also	30
saEvtChannelOpenAsync(), saEvtDispatch()	
	35

SERVICE AVAILABILITY" FORUM

3.6.3 saEvtChannelClose()		
Prototype		
SaAisErrorT saEvtChannelClose(5	
SaEvtChannelHandleT channelHandle		
);		
Parameters	10	
<i>channelHandle</i> - [<i>in</i>] The handle of the event channel to close. The <i>channelHandle</i> parameter must have been obtained previously in an invocation of <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenCallback()</i> . The <i>SaEvtChannelHandleT</i> type is defined in Section 3.4.1.3 on page 18.	15	
Description		
This API function closes the event channel which is designated by <i>channelHandle</i> and which was opened by an earlier invocation of <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> .	20	
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 <i>channelHandle</i> . 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 <i>saEvtEventAllocate()</i> or <i>saEvtEventDeliverCallback()</i> (and that have not yet been freed by <i>saEvtEventFree()</i>) and frees memory allocated	25	
for the process by saEvtEventAttributesGet() that has not yet been freed by saEvtEventPatternFree().	30	
This call cancels all pending callbacks that refer directly or indirectly to the handle <i>channelHandle</i> . 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 <i>saEvtChannelClose()</i> function completes successfully, and no process has the event channel open any longer, and the deletion of the event channel nel was pending as a result of invoking the <i>saEvtChannelUnlink()</i> function, the event channel is deleted immediately.	35	
The deletion (unlink) of an event channel (see Section 3.6.4) frees all resources allo- cated by the Event Service for it, such as published events with nonzero retention time.	40	



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 pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>channelHandle</i> 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()	

SERVICE AVAILABILITY" FORUM

3.6.4 saEvtChannelUnlink()	1
Prototype	
SaAisErrorT saEvtChannelUnlink(5
SaEvtHandleT evtHandle,	
const SaNameT *channelName	
);	10
Parameters	-
<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Every Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.	ent 15
<i>channelName</i> - [<i>in</i>] A pointer to the name of the event channel that is to be unlink The <i>SaNameT</i> type is defined in [1].	əd.
Description	20
This function deletes from the cluster an event channel designated by the name to which <i>channelName</i> points.)
After successful completion of the invocation:	25
 The name to which channelName points is no longer valid, that is, any invocat of a function of the Event Service API that uses this event channel name retur an error unless an event channel is re-created with this name. The event cha nel is re-created by specifying in an saEvtChannelOpen() call or an saEvtChannelOpenAsync() call the SA_EVT_CHANNEL_CREATE flag and t same name of the event channel to be unlinked. This way, a new instance of t event channel is created while the old instance of the event channel is possit not yet finally deleted. 	ion rns in- he 30 the
 Note that this behavior is similar to the way POSIX treats files. If no process has the event channel open when <i>saEvtChannelUnlink()</i> is invok the event channel is immediately deleted. 	ed, 35
 Any process that has the event channel open can still continue to access it. Do tion of the event channel will occur when the last saEvtChannelClose() operat is performed. 	ion
Note that the only way to delete an existing event channel from the cluster namespace is by invoking <i>saEvtChannelUnlink()</i> on the event channel.	40



The deletion of an event channel frees all resources allocated by the Event Service for it, such as published events with nonzero retention time.	1
This API can be invoked by any process, and the invoking process need not be the creator or opener of the event channel.	5
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.	10
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 pro- cess may retry later.	15
SA_AIS_ERR_BAD_HANDLE - The handle <i>evtHandle</i> is invalid, since it is corrupted, uninitialized, or has already been finalized.	
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	20
SA_AIS_ERR_NOT_EXIST - The event channel identified by the name to which <i>channelName</i> 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.	25
See Also	
saEvtInitialize(), saEvtChannelClose()	
	30

SERVICE AVAILABILITY" FORUM

3.7 Event Operations	1
3.7.1 saEvtEventAllocate()	
Prototype	5
SaAisErrorT saEvtEventAllocate(
SaEvtChannelHandleT channelHandle,	
SaEvtEventHandleT *eventHandle	10
);	
Parameters	
<i>channelHandle</i> - [<i>in</i>] The handle of the event channel on which the event is to be pub- lished. The <i>channelHandle</i> parameter must have been obtained previously in an invo- cation of <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenCallback()</i> . The <i>SaEvtChannelHandleT</i> type is defined in Section 3.4.1.3 on page 18.	15
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 invok- ing this function. The Event Service will assign the value of the handle when this func- tion returns successfully. The <i>SaEvtEventHandleT</i> type is defined in Section 3.4.1.2 on page 18.	20 25
Description	20
The <i>saEvtEventAllocate()</i> function allocates memory for the event header and initial- izes all event attributes to default values, as described in Section 3.4.5.5 on page 21. The event allocated by <i>saEvtEventAllocate()</i> must be freed by invoking <i>saEvtEventFree()</i> .	30
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 <i>channelHandle</i> was opened.	35
Return Values	50
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.	40



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.	1
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	5
SA_AIS_ERR_BAD_HANDLE - The handle <i>channelHandle</i> 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. 	10
 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.	15
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).	20
SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_PUBLISHER flag was not set when the instance of the event channel identified by <i>channelHandle</i> 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.	25
See Also	
saEvtEventFree(), saEvtEventPublish()	
	30



3.7.2 saEvtEventFree()		
Prototype		
SaAisErrorT saEvtEventFree(SaEvtEventHandleT eventHandle	5	
) ; Parameters	10	
<i>eventHandle</i> - [<i>in</i>] The handle of the event whose memory can now be freed by the Event Service. The <i>SaEvtEventHandleT</i> type is defined in Section 3.4.1.2 on page 18.	10	
Description	15	
The function is used to free events allocated by <i>saEvtEventAllocate()</i> or obtained from <i>saEvtEventDeliverCallback()</i> .		
The <i>saEvtEventFree()</i> 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 <i>eventHandle</i> . If the event was allocated by the Event Service during a prior invocation of <i>saEvtEventDeliverCallback()</i> , <i>saEvtEventFree()</i> also gives the Event Service permission to deallocate the event data.	20	
Return Values	25	
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.	30	
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 pro- cess may retry later.	35	
SA_AIS_ERR_BAD_HANDLE - The handle <i>eventHandle</i> 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. 	40	



	 The corresponding event channel has already been closed. The handle <i>evtHandle</i> that was passed to the <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> functions has already been finalized. 	1
	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	5
	See Also	
	SaEvtEventDeliverCallbackT, saEvtEventAllocate(), saEvtChannelOpen(), saEvtChannelOpenAsync()	10
3.7.3	saEvtEventAttributesSet()	
	Prototype	15
	SaAisErrorT saEvtEventAttributesSet(15
	SaEvtEventHandleT eventHandle,	
	const SaEvtEventPatternArrayT *patternArray,	
	SaEvtEventPriorityT priority,	20
	SaTimeT retentionTime,	
	const SaNameT *publisherName	
);	25
	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.	30
	<i>patternArray</i> - [<i>in</i>] 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 <i>SaEvtEventPatternArrayT</i> type is defined in Section 3.4.5.2 on page 20.	
	<i>priority</i> - [<i>in</i>] The priority of the event. The <i>SaEvtEventPriorityT</i> type is defined in Sec- tion 3.4.5.3 on page 21.	35
	<i>retentionTime</i> - [<i>in</i>] The duration for which the event will be retained. The <i>SaTimeT</i> type is defined in [1].	40
	<i>publisherName</i> - [<i>in</i>] A pointer to the name of the publisher of the event. The <i>SaNameT</i> type is defined in [1].	



5

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 10 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 theSA_EVT_CHANNEL_PUBLISHER flag was not set when either saEvtChannelOpen()or saEvtChannelOpenAsync() was called to open the instance of the event channel15on which the handle eventHandle was obtained.

Return Values

SA_AIS_OK - The function completed successfully.	20
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.	25
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>eventHandle</i> is invalid, due to one or more of the reasons below:	30
 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. 	35
 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.	40
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.	40



SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory).	1
SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_PUBLISHER flag was not set when either <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> was called to open the instance of the event channel on which the handle <i>eventHandle</i> was obtained.	5
SA_AIS_ERR_TOO_BIG - One or more of the following conditions occur:	
 The value of <i>patternArray->patternsNumber</i> 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. 	10
 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. 	15
 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.	20
See Also	
saEvtEventAllocate(),	25



3.7.4 s	3.7.4 saEvtEventAttributesGet()	
	Prototype	
	SaAisErrorT saEvtEventAttributesGet(5
	SaEvtEventHandleT eventHandle,	
	SaEvtEventPatternArrayT *patternArray,	
	SaEvtEventPriorityT *priority,	10
	SaTimeT *retentionTime,	10
	SaNameT *publisherName,	
	SaTimeT *publishTime,	
	SaEvtEventIdT *eventId	15
);	
	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.	20
	<i>patternArray</i> - [<i>in/out</i>] A pointer to a structure that describes the event pattern array and the number of patterns to be retrieved. The <i>SaEvtEventPatternArrayT</i> type is defined in Section 3.4.5.2 on page 20.	25
	If the caller sets <i>patternArray->patterns</i> to NULL, the Event Service ignores the <i>patternArray->allocatedNumber</i> field and allocates memory for <i>patternArray->patterns</i> and each <i>patternArray->patterns</i> [<i>i</i>]. <i>pattern</i> (where <i>i</i> is an index to one of the patterns). The Event Service sets the fields <i>patternArray->patternsNumber</i> , <i>patternArray->patterns</i> ,	30
	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.	35
	Alternatively, the invoking process can allocate the memory to retrieve all event pat- terns and set the fields <i>patternArray->allocatedNumber</i> , <i>patternArray->patterns</i> , <i>patternArray->patterns</i> [<i>i</i>]. <i>allocatedSize</i> , and <i>patternArray->patterns</i> [<i>i</i>]. <i>pattern</i> accordingly. In this case, these fields are <i>in</i> parame-	40



ters and will not be modified by the Event Service.	1
The Event Service copies the patterns into the successive entries of patternArray->patterns, starting with the first entry and continuing until all event pat-	
terns are copied. If <i>patternArray->allocatedNumber</i> is smaller than the number of	
event patterns, or the size of the buffer allocated for one of the patterns is smaller	5
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 <i>patternArray->patterns[i</i>]. <i>pattern</i> point were changed or not by the Event Service. Regardless of whether such an error occurs, the Event	
Service sets the <i>patternArray->patternsNumber</i> and	10
patternArray->patterns[i].patternSize fields for all	
patternArray->allocatedNumber individual patterns to indicate the number of event	
patterns and the size of each pattern.	
<i>priority</i> - [<i>out</i>] A pointer to the priority of the event. The SaEvtEventPriorityT type is	15
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].	20
<i>publisherName</i> - [<i>out</i>] A pointer to the name of the publisher of the event. The	
SaNameT type is defined in [1].	
nublishTime. [out] A pointer to the time of which the publisher published the event	
<i>publishTime</i> - [<i>out</i>] A pointer to the time at which the publisher published the event. The <i>SaTimeT</i> type is defined in [1].	25
eventId - [out] A pointer to the event identifier. The SaEvtEventIdT type is defined in	
Section 3.4.5.4 on page 21.	
Description	30
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 Ser-	
vice does not return the corresponding <i>out</i> value.	35
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 <i>saEvtEventAttributesSet()</i> function after the event has	40
been allocated or received.	
Only if this API is invoked on a received event, the attributes	

SERVICE AVAILABILITY FORUM

"publish time" and	1
"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 allo- cated the event or the attributes set by a prior invocation of the <i>saEvtEventAttributesSet()</i> function.	5
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 <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> was called to open the instance of the event channel on which the handle <i>eventHandle</i> was obtained.	10
Return Values	4.5
SA_AIS_OK - The function completed successfully.	15
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.	20
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	25
SA_AIS_ERR_BAD_HANDLE - The handle <i>eventHandle</i> is invalid, due to one or more of the reasons below:	25
 It is corrupted, was not obtained with the saEvtEventAllocate() or saEvtEventDeliverCallback() functions, or saEvtEventFree() has already been invoked for eventHandle. 	30
 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.	35
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 <i>allocatedNumber</i> in the structure to which <i>patternArray</i> 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.	40



	This return value applies only if the <i>patterns</i> pointer as an <i>in</i> parameter in the struc- ture to which <i>patternArray</i> points is not NULL.	1
	SA_AIS_ERR_ACCESS - Neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> was called to open the instance of the event channel on which the handle <i>eventHandle</i> was obtained.	5
	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	10
	See Also	10
	saEvtEventPatternFree(), SaEvtEventDeliverCallbackT, saEvtEventAllocate(), saEvtEventFree(), saEvtChannelOpen(), saEvtChannelOpenAsync(), saEvtEventAttributesSet(), saEvtFinalize()	15
3.7.5	saEvtEventPatternFree()	
	Prototype	
	SaAisErrorT saEvtEventPatternFree(20
	SaEvtEventHandleT eventHandle,	
	SaEvtEventPatternT *patterns	
);	25
	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.	30
	<i>patterns</i> - [<i>in</i>] A pointer to the memory that was allocated by the Event Service library in the <i>saEvtEventAttributesGet()</i> function and is to be deallocated. The <i>SaEvtEventPatternT</i> type is defined in Section 3.4.5.1 on page 20.	
	Description	35
	This function frees the memory to which <i>patterns</i> points and which was allocated by the Event Service library in a previous call to the <i>saEvtEventAttributesGet()</i> function.	
	For details, refer to the description of the <i>patternArray</i> parameter of the <i>saEvtEventAttributesGet()</i> function.	40



	Return Values	1
	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.	5
	SA_AIS_ERR_BAD_HANDLE - The handle <i>eventHandle</i> 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. 	10
	 The corresponding event channel has already been closed. 	
	 The handle evtHandle that was passed to the saEvtChannelOpen() or saEvtChannelOpenAsync() functions has already been finalized. 	15
	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	20
	saEvtEventAttributesGet()	
3.7.6	saEvtEventDataGet()	05
	Prototype	25
	SaAisErrorT saEvtEventDataGet(
	SaEvtEventHandleT eventHandle,	30
	void *eventData,	50
	SaSizeT *eventDataSize	
);	
	Parameters	35
	<i>eventHandle</i> - [<i>in</i>] The handle to the event that was delivered by <i>saEvtEventDeliverCallback()</i> . The <i>SaEvtEventHandleT</i> type is defined in Section 3.4.1.2 on page 18.	40
	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.	

SAI-AIS-EVT-B.03.01 Section 3.7.6



5

10

20

30

35

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 the15SA_EVT_CHANNEL_SUBSCRIBER flag was not set when either15saEvtChannelOpen() or saEvtChannelOpenAsync() was called to open the instance15of the event channel on which eventHandle was obtained.15

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. 40

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.



5

15

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 10 this cluster node because it is not a member node.

See Also

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

3.7.7 SaEvtEventDeliverCallbackT

Prototype	
typedef void(*SaEvtEventDeliverCallbackT)(20
SaEvtSubscriptionIdT subscriptionId,	
SaEvtEventHandleT eventHandle,	
SaSizeT eventDataSize	25

);

Parameters

30 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	1
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 <i>saEvtDispatch()</i> on the han- dle <i>evtHandle</i> that was specified in the corresponding <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> call to obtain the channel handle to the instance of the event channel on which <i>eventHandle</i> was obtained.	5
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.	10
The process may invoke <i>saEvtEventAttributesGet()</i> to obtain the attributes associ- ated with the event and <i>saEvtEventDataGet()</i> to obtain the data associated with the event.	15
It is the responsibility of the process to free the event by invoking the saEvtEventFree() function.	
The validity of the <i>eventHandle</i> parameter is not limited to the scope of this callback function.	20
To ensure that the <i>subscriptionId</i> 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 <i>subscriptionIds</i> that are unique across all open event channels when calling the <i>saEvtEventSubscribe()</i> function.	25
Return Values	
None	30
See Also	
saEvtEventAttributesGet(), saEvtEventDataGet(), saEvtEventSubscribe(), saEvtChannelOpen(), saEvtChannelOpenAsync(), saEvtEventFree(), saEvtDispatch()	35



3.7.8 saEvtEventPublish()	
Prototype	
SaAisErrorT saEvtEventPublish(5
SaEvtEventHandleT eventHandle,	
const void *eventData,	
SaSizeT eventDataSize,	10
SaEvtEventIdT *eventId	10
);	
Parameters	15
<i>eventHandle</i> - [<i>in</i>] The handle of the event that is to be published. The event must have been allocated by <i>saEvtEventAllocate()</i> or obtained from <i>saEvtEventDeliverCallback()</i> . If changes are required, the patterns must have been	15
set by a prior call to <i>saEvtEventAttributesSet()</i> . The <i>SaEvtEventHandleT</i> type is defined in Section 3.4.1.2 on page 18.	20
eventData - [<i>in</i>] 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 <i>eventData</i> when <i>saEvtEventPublish()</i> returns.	25
<i>eventDataSize</i> - [<i>in</i>] The number of bytes in the buffer to which <i>eventData</i> points. This parameter is ignored if <i>eventData</i> is NULL. The <i>SaSizeT</i> type is defined in [1].	
<i>eventId</i> - [<i>out</i>] A pointer to an identifier of the event. The <i>SaEvtEventIdT</i> type is defined in Section 3.4.5.4 on page 21.	30
Description	
If the <i>saEvtEventPublish()</i> function completes successfully, it publishes an event on the event channel on which the event identified by <i>eventHandle</i> was either allocated by invoking the <i>saEvtEventAllocate()</i> function or obtained from the <i>saEvtEventDeliverCallback()</i> function, and it returns the event identifier in the field to which <i>eventId</i> points. The event to be published consists of a standard set of attributes (the event header) and an optional data part.	35 40
Before an event is published, the publisher process may invoke the saEvtEventAttributesSet() function to set the writeable event attributes. The pub-	



lished event is delivered to subscribers whose subscription filters match the event patterns.	1
When the Event Service publishes an event, it automatically sets the following read- only event attributes into the published event:	5
Event publish time	
Event identifier	
In addition to the event attributes, a process can supply values for the <i>eventData</i> and <i>eventDataSize</i> parameters for publication as part of the event.	10
The event attributes and the event data of the event identified by <i>eventHandle</i> are not affected by this API function.	
The invocation of <i>saEvtEventPublish()</i> copies the event attributes and the event data into internal memory of the Event Service. The invoking process can free the event by invoking <i>saEvtEventFree()</i> after <i>saEvtEventPublish()</i> returns.	15
This function fails with the SA_AIS_ERR_ACCESS error code if the SA_EVT_CHANNEL_PUBLISHER flag was not set when either <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> was called to open the instance of the event channel on which the handle <i>eventHandle</i> was obtained.	20
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.	25
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.	30
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>eventHandle</i> is invalid, due to one or more of the reasons below:	35
 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. 	40
 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.	1
	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).	5
	SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_PUBLISHER flag was not set when either <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> was called to open the instance of the event channel on which the handle <i>eventHandle</i> was obtained.	10
	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.	15
	See Also	
	saEvtEventAttributesSet(),	20
3.7.9	saEvtEventSubscribe()	
	Prototype	05
	SaAisErrorT saEvtEventSubscribe(25
	SaEvtChannelHandleT channelHandle,	
	const SaEvtEventFilterArrayT *filters,	
	SaEvtSubscriptionIdT subscriptionId	30
);	

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

35



<i>saEvtEventSubscribe()</i> returns. The <i>SaEvtEventFilterArrayT</i> type is defined in Sec- tion 3.4.6.3 on page 23.	1
<i>subscriptionId</i> - [<i>in</i>] An identifier that uniquely identifies a particular subscription on the instance of the opened event channel designated by <i>channelHandle</i> . This identifier is used as a parameter of the <i>saEvtEventDeliverCallback()</i> function. The <i>SaEvtSubscriptionIdT</i> type is defined in Section 3.4.2 on page 18.	5
Description	
The <i>saEvtEventSubscribe()</i> function enables a process to subscribe for events on an event channel by registering one or more filters on that event channel.	10
Events are delivered by the invocation of the <i>saEvtEventDeliverCallback()</i> callback function, which must have been supplied when the process called the <i>saEvtInitialize()</i> function.	15
The memory associated with the filters is not deallocated by the <i>saEvtEventSubscribe()</i> function. It is the responsibility of the invoking process to de-allocate the memory when the <i>saEvtEventSubscribe()</i> function returns.	
For a particular subscription, the filter patterns cannot be modified. To change the fil- ter 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 <i>saEvtEventUnsubscribe()</i> function.	20
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 <i>channelHandle</i> was opened.	25
Return Values	
SA_AIS_OK - The function completed successfully.	30
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.	35
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>channelHandle</i> is invalid, due to one or both of the reasons below:	40



 It is corrupted, was not obtained with the saEvtChannelOpen() or saEvtChannelOpenCallback() functions, or the corresponding event channel has already been closed. 	1
 The handle evtHandle that was passed to the saEvtChannelOpen() or saEvtChannelOpenAsync() functions has already been finalized. 	5
SA_AIS_ERR_INIT - The previous invocation of <i>saEvtInitialize()</i> to initialize the Event Service was incomplete, since the <i>saEvtEventDeliverCallback()</i> callback function is missing.	
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	10
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).	15
SA_AIS_ERR_EXIST - A subscription using the same <i>subscriptionId</i> already exists on the instance of the opened event channel designated by <i>channelHandle</i> .	
SA_AIS_ERR_ACCESS - The SA_EVT_CHANNEL_SUBSCRIBER flag was not set when the instance of the event channel identified by <i>channelHandle</i> was opened.	20
SA_AIS_ERR_TOO_BIG - The value of <i>filtersNumber</i> in the structure to which the <i>filters</i> 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.	25
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	30
SaEvtEventDeliverCallbackT, saEvtEventUnsubscribe(), saEvtEventDataGet(), saEvtEventAttributesGet()	
	35



3.7.10 saEvtEventUnsubscribe()	1
Prototype	
SaAisErrorT saEvtEventUnsubscribe(5
SaEvtChannelHandleT channelHandle,	
SaEvtSubscriptionIdT subscriptionId	
);	10
Parameters	
<i>channelHandle</i> - [<i>in</i>] The handle of the event channel on which the subscriber is requesting the Event Service to delete the subscription. The <i>channelHandle</i> parameter must have been obtained previously in an invocation of <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenCallback()</i> . The <i>SaEvtChannelHandleT</i> type is defined in Section 3.4.1.3 on page 18.	15
<i>subscriptionId</i> - [<i>in</i>] An identifier that uniquely identifies a particular subscription on the instance of the opened event channel designated by <i>channelHandle</i> . The <i>SaEvtSubscriptionIdT</i> type is defined in Section 3.4.2 on page 18.	20
Description	
The <i>saEvtEventUnsubscribe()</i> 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 <i>channelHandle</i> and <i>subscriptionId</i> and must have been set up by a previous invocation of the <i>saEvtEventSubscribe()</i> function for the same pair.	25
It is an error if the value of the <i>subscriptionId</i> parameter does not match a previously registered subscription on the instance of the opened event channel identified by <i>channelHandle</i> .	30
The <i>saEvtEventUnsubscribe()</i> 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 <i>channelHandle</i> and <i>subscriptionId</i> .	35
A process that wants to modify a subscription without losing any events must estab- lish the new subscription before removing the existing subscription.	
Return Values	40
SA_AIS_OK - The function completed successfully.	



5

10

20

25

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 ¹⁵ *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	30
SaAisErrorT saEvtEventRetentionTimeClear(
SaEvtChannelHandleT channelHandle,	
const SaEvtEventIdT eventId	35

);

Parameters

channelHandle - [in] The handle of the event channel on which the event has been40published. The handle channelHandle must have been obtained previously in an40invocation of saEvtChannelOpen() or saEvtChannelOpenCallback(). The50SaEvtChannelHandleT type is defined in Section 3.4.1.3 on page 18.50



<i>eventId</i> - [<i>in</i>] The identifier of the event. The <i>SaEvtEventIdT</i> type is defined in Section 3.4.5.4 on page 21.	1
Description	
The <i>saEvtEventRetentionTimeClear()</i> function is used to clear the retention time of the published event designated by <i>eventId</i> . 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.	5
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 <i>saEvtChannelOpen()</i> or <i>saEvtChannelOpenAsync()</i> functions was called to open the instance of the event channel identified by <i>channelHandle</i> .	15
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.	20
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.	25
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	25
SA_AIS_ERR_BAD_HANDLE - The handle <i>channelHandle</i> is invalid, due to one or both of the reasons below:	30
 It is corrupted, was not obtained with the saEvtChannelOpen() or saEvtChannelOpenCallback() functions, or the corresponding event channel has already been closed. 	50
 The handle evtHandle that was passed to the saEvtChannelOpen() or saEvtChannelOpenAsync() functions has already been finalized. 	35
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this error is returned if <i>eventId</i> is not a valid event identifier.	
SA_AIS_ERR_NOT_EXIST - The event specified by <i>eventId</i> does not exist.	40
SA_AIS_ERR_ACCESS - Neither the SA_EVT_CHANNEL_PUBLISHER flag nor the SA_EVT_CHANNEL_SUBSCRIBER flag was set when one of the	40



	saEvtChannelOpen() or saEvtChannelOpenAsync() functions was called to open the instance of the event channel identified by channelHandle.	1
	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	5
	See Also	
	saEvtEventPublish(), SaEvtEventDeliverCallbackT	
3.8 L	imit Fetch API	10
3.8.1 :	saEvtLimitGet()	
	Prototype	15
	SaAisErrorT saEvtLimitGet(
	SaEvtHandleT evtHandle,	
	SaEvtLimitIdT limitId,	20
	SaLimitValueT *limitValue	20
);	
	Parameters	25
	<i>evtHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saEvtInitialize()</i> function and which designates this particular initialization of the Event Service. The <i>SaEvtHandleT</i> type is defined in Section 3.4.1.1 on page 18.	25
	<i>limitId</i> - [<i>in</i>] The Event Service limit whose implementation-specific value is to be queried. The limits are defined in the <i>SaEvtLimitIdT</i> type in Section 3.4.8 on page 27.	30
	<i>limitValue</i> - [<i>out</i>] Pointer to the current value of the limit specified in <i>limitId</i> . For details regarding this type, refer to the SA Forum Overview document ([1]).	25
	Description	35
	This function enables a user application to obtain the current implementation-specific value of an Event Service limit. The <i>limitId</i> 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 <i>limitValue</i> .	40



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 pro- cess may retry later.
SA_AIS_ERR_BAD_HANDLE - The handle <i>evtHandle</i> 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 <i>limitId</i> parameter contains an invalid value.
The <i>limitValue</i> pointer is NULL.
SA_AIS_ERR_VERSION - The invoked function is not supported in the version spec- ified 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()



5
10
15
20
25
30
35



4	Event Service UML Information Model	1
	The Event Service information model is described in UML and has been organized in a UML class diagram.	F
	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]).	5
	The Event Service UML class diagram has one class, which shows the contained attributes.	10

4.1 DN Format for the Event Service UML Class

Table 3 DN Formats for Objects of the Event Service Class		
DN Format for Objects of the Class		
"safChnl=,* "	20	
	DN Format for Objects of the Class	

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]).

35

25



	< <runtime>></runtime>
	SaEvtChannel
safChnl : SaStringT	[1]{RDN, RUNTIME, CACHED}
	ionTimestamp : SaTimeT [1]{RUNTIME, CACHED
	peners : SaUint32T [1]{RUNTIME}
	Subscribers : SaUint32T [1]{RUNTIME}
	ublishers : SaUint32T [1]{RUNTIME}
	<pre>RetainedEvents : SaUint32T [1]{RUNTIME} iventsEventCount : SaUint32T [1]{RUNTIME}</pre>

25

1

5

10

15

30

35



5	Event Service Administration API	1
	The Event Service has no administration interface at the time of publication of this specification.	5
		10
		15
		20
		25
		30
		35
		40



5
10
15
20
25
30
35



Alarms and Notifications	
The Event Service does not issue any alarms and notifications at the time of publica- tion of this specification.	
	The Event Service does not issue any alarms and notifications at the time of publica-



5
10
15
20
25
30
35



7	Event Service Management Interface	1
	Currently, an SNMP MIB interface is defined for the Event Service. Other manage- ment access methods to the Event Service may be added in future versions of this specification.	5
7.1	Event Service MIB (SAF-EVT-SVC-MIB)	
	The Event Service MIB contains the single read-only table <i>saEvtChannelTable</i> , 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.	10
	This table mimics the UML runtime object class <i>SaEvtChannel</i> , as described in Sec- tion 4.2 in terms of the objects contained in the table.	15
	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.	00
	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 .	20
		25



5
10
15
20
25
30
35

SERVICE AVAILABILITY FORUM

Index of Definitions

Α

at most once delivery property 14 attributes see event attributes

в

best effort delivery property 14

С

channel *see* event channels completeness property 14

D

data see event data

Е

event attributes 13 event channels see also events definition 13 at most once delivery property 14 best effort delivery property 14 completeness property 14 filter types 24 filtering 15 filters 22 ordering 14 pattern array 22 pattern matching algorithms 24 patterns 19 persistence property 15 publisher 13 retention time property 14 subscriber 13 subscription 18 event completeness property 14 event data 13 event filtering 15 event filters 22 event header 13 event id 22 event ordering 14 event pattern array 22 event patterns 19 event priority 14, 22 event publish time 22 event publisher name 22 event retention time 14, 22 events see also event channels definition 13 attributes 13 data 13 header 13 id 22 lost event 25 priority 22 publish time 22

published 13 publisher name 22 retention time 22	1
F filter types 24 filtering <i>see</i> event filtering filters <i>see</i> event filters	5
H header <i>see</i> event header	
I id <i>see</i> event id identification <i>see</i> event id	10
L lost event 25	
O ordering <i>see</i> event ordering	15
P pattern array see event pattern array pattern matching algorithms 24 patterns see event patterns persistence property 15	10
priority see event priority publish time see event publish time published event 13 publisher 13 publisher name see event publisher name	20
R retention time see event retention time	25
S subscriber 13 subscription 18	20
	30
	35
	40



5
10
15
20
25
30
35