Service Availability™ Forum Application Interface Specification

Message Service SAI-AIS-MSG-B.02.01

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

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

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

The Service Availability™ Specification(s) (the "Specification") found at the URL http://www.saforum.org (the "Site") is generally made available by the Service Availability Forum (the "Licensor") for use in developing products that are compatible with the standards provided in the Specification. The terms and conditions, which govern the use of the Specification are set forth in this agreement (this "Agreement").

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

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

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

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

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

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

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

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

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

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

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

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

5. OWNERSHIP OF SPECIFICATION AND COPYRIGHTS. The Specification and all worldwide copyrights therein are the exclusive property of Licensor. You may not remove, obscure, or alter any copyright or other proprietary rights notices that are in or on the copy of the Specification you download. You must reproduce all such notices on all copies of the Specification you make. Licensor may make changes to the Specification, or to items referenced

therein, at any time without notice. Licensor is not obligated to support or update the Specification.

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

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

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

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

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

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

Table of Contents Message Service

1 Document Introduction

1.1 Document Purpose

This document defines the Message 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.

Typically, the Service Availability™ Forum Application Interface Specification will be used in conjunction with the Service Availability™ Forum Hardware Interface Specification (HPI) and with the Service Availability™ Forum System Management Specification.

1.2 AIS Documents Organization

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

1.3 History

Previous releases of the Message Service specification:

(1) SAI-AIS-MSG-A.01.01

(2) SAI-AIS-MSG-B.01.01

This section presents the changes of the current release, SAI-AIS-MSG-B.02.01, with respect to the SAI-AIS-MSG-B.01.01 release. Editorial changes are not mentioned here.

1.3.1 New Topics

- Section 3.3.12 on page 29 for the *saMsgMessageCapacityStatusT* data type.
- Section 3.3.13 on page 30 for the *SaMsgStateT* data type.
- Section 3.5.5 on page 45 for the *saMsgQueueRetentionTimeSet()* function.
- Section 3.6.8 on page 59 for the *saMsgQueueGroupNotificationFree()* function, which is used to free memory allocated by the Message Service library in the *saMsgQueueGroupTrack()* function.

- Section 3.7.4 on page 70 for the *saMsgMessageDataFree()* function, which is used to free memory allocated by the Message Service library in the *saMsgMessageGet()* or *saMsgMessageSendReceive()* functions.
- Chapter 4 Alarms and Notifications on page 81.

1.3.2 Clarifications

- The behavior of the local equal load queue distribution is clarified on page 18.
- Section 3.1.5 on page 19 clarifies that while a process has a message queue open, the message queue cannot be opened again by the same process or by any other process.
- Section 3.1.6 on page 20 clarifies that the SA_AIS_ERR_QUEUE_FULL error code is only delivered for a process sending messages to a queue if the process requested an acknowledgement for its send operation.
- The description of the SA_MSG_QUEUE_RECEIVE_CALLBACK flag in *SaMsgQueueOpenFlagsT (*see Section 3.3.6 on page 23) clarifies that a message can be retrieved by calling *saMsgMessageGet()* during the invocation of the *saMsgMessageReceivedCallback()* callback function.
- Section 3.3.11 on page 28 clarifies the setting of the *senderName* in the *SaMsgMessageT* structure.
- The description of the functions *saMsgMessageSendAsync()* in Section 3.7.1 on page 61, *saMsgMessageReplyAsync()* in Section 3.8.2 on page 77, and *SaMsgMessageDeliveredCallbackT* in Section 3.7.2 on page 64 clarify that a process may deallocate the memory for the data in the message buffer passed to *saMsgMessageSendAsync() or saMsgMessageReplyAsync()* also during the invocation of the corresponding *saMsgMessageDeliveredCallback()* function.

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
SaMsgMessageDeliveredCallbackT	SA_AIS_ERR_BAD_HANDLE SA_AIS_ERR_INVALID_PARAM SA_AIS_ERR_BAD_FLAGS	new
saMsgMessageGet()	SA_AIS_ERR_NO_MEMORY	new
saMsgMessageReply() *saMsgMessageReplyAsync()*	SA_AIS_ERR_BUSY SA_AIS_ERR_NOT_EXIST	clarified
saMsgMessageReplyAsync()	SA_AIS_ERR_INIT	new
saMsgQueueGroupCreate()	SA_AIS_ERR_INVALID_PARAM	extended
saMsgQueueGroupTrack()	SA_AIS_ERR_INIT SA_AIS_ERR_NO_SPACE	clarified
SaMsgQueueGroupTrackCallbackT	SA_AIS_ERR_BAD_HANDLE SA_AIS_ERR_INVALID_PARAM SA_AIS_ERR_BAD_FLAGS	new
saMsgQueueOpen() and *saMsgQueueOpenAsync()*	SA_AIS_ERR_INVALID_PARAM SA_AIS_ERR_EXIST	extended
saMsgQueueOpen() and *saMsgQueueOpenAsync()*	SA_AIS_ERR_BUSY	clarified
SaMsgQueueOpenCallbackT	SA_AIS_ERR_BAD_HANDLE SA_AIS_ERR_INVALID_PARAM SA_AIS_ERR_BAD_FLAGS	new
SaMsgQueueOpenCallbackT	SA_AIS_ERR_BUSY	clarified

1.3.4 Other Changes

- The name spaces for message queues and message queue groups are now distinct. This is explained in Section 3.1.3 on page 17.
- The definition of the enum values SA_MSG_QUEUE_GROUP_NO_CHANGE and SA_MSG_QUEUE_GROUP_STATE_CHANGED in Section 3.3.10.1 on page 26 have been changed. The meaning of the SA_MSG_QUEUE_GROUP_NO_CHANGE enum has been clarified. The SA_MSG_QUEUE_GROUP_STATE_CHANGED enum value is no longer used and is now reserved for future use.

- The description of the *creationAttributes* parameter in the *saMsgQueueOpen()* and *saMsgQueueOpenAsync()* functions (see Section 3.5.1 on page 36) states now that *retentionTime* is ignored when the *creationAttributes* parameter is not NULL, and the message queue already exists.
- Memory allocated by the Message Service library in the *saMsgQueueGroupTrack()* function (see Section 3.6.5 on page 53) must now be freed by invoking the *saMsgQueueGroupNotificationFree()* function (see Section 3.6.8 on page 59).
- Memory allocated by the Message Service library in the functions *saMsgMessageGet()* (see Section 3.7.3 on page 67) or *saMsgMessageSendReceive() (*see Section 3.8.1 on page 73) must now be freed by invoking the *saMsgMessageDataFree()* function (see Section 3.7.4 on page 70*).*

1.4 References

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

[1] Service Availability™ Forum, Application Interface Specification, Overview, SAI-Overview-B.02.01

[2] Service Availability™ Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.01.01

[3] Service Availability™ Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.02.01

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

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

1.5 How to Provide Feedback on the Specification

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

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

1.6 How to Join the Service Availability™ Forum

The Promoter Members of the Forum require that all organizations wishing to participate in the Forum complete a membership application. Once completed, a representative of the Service Availability™ Forum will contact you to discuss your

membership in the Forum. The Service Availability™ Forum Membership Application can be completed online by following the pertinent links provided on the Forum's website (**http://www.saforum.org**).

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

1.7 Additional Information

1.7.1 Member Companies

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

1.7.2 Press Materials

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

2 Overview

This specification defines the Message Service within the Application Interface Specification (AIS).

2.1 Message Service

The Message Service specifies a buffered message passing system based on the concept of a message queue for processes on the same or on different nodes[1]. Messages are written to and read from message queues. A single message queue permits a multipoint-to-point communication. Message queues are persistent or non-persistent. The Message Service must preserve messages that have not yet been consumed when the message queue is closed.

Processes sending messages to a message queue are unaware that the process, which was originally processing these messages, has been replaced by another process acting as a standby in case the original process fails or switches over.

Message queues can be grouped together to form message queue groups. Message queue groups permit multipoint-to-multipoint communication. They are identified by logical names, so that a process is unaware of the number of message queues and of the physical location of the message queues to which it is communicating. The sender addresses message queue groups using the same mechanisms that it uses to address single message queues. The message queue groups can be used to distribute messages among message queues pertaining to the message queue group. Regardless of the number of message queues to which messages are distributed, the message queue group remains accessible under the same name.

Message queue groups can be used to maintain transparency of the sender process to faults in the receiver processes, represented by the message queues in the message queue groups. The sender process communicates with the message queue group. If a receiver process fails, the sender process continues to communicate with the message queue group and is unaware of the fault, because it continues to obtain service from the other receiver processes.

With message queues, the Message Service uses the model of *n* senders to *one* receiver whereas, with message queue groups, the Message Service uses the model of *m* senders to *n* receivers.

1. The term “node” is used in this document in the sense of a “member node”, as defined in the Cluster Membership Service specification.

3 SA Message Service API

3.1 Message Service Model

3.1.1 Messages

A **message** is a byte array of a certain size. In addition to the data contained in the byte array, a message also has a **type**, **version**, **priority,** and **sender name**. The contents of a message are opaque to the Message Service.

3.1.2 Message Queues

A **message queue** is a software abstraction for buffering messages. A message queue consists of a collection of separate data areas that are used to store messages of different priorities. These data areas are called the **priority areas** of the message queue. Each priority area holds messages of the same priority. The priority areas of a message queue have individual sizes.

A message queue is global to a cluster and is identified by a unique name in the name space of all message queues. A message queue can be configured either statically or dynamically. The static configuration interfaces are not part of this standard.

3.1.3 Message Queue Groups

A **message queue group** is a collection of message queues that are addressed as a single entity. A message queue group is global to a cluster and is identified by a unique name. Message queues and message queue groups have distinct name spaces. A message queue can be a member of more than one message queue group. If a message queue is removed from the cluster, it is automatically removed from all message queue groups it is a member of.
The names, properties and members of a message queue group are specified either statically or dynamically. The static configuration interfaces are not defined by this specification.

Messages sent to a message queue group are directed to one or more of its member message queues. In a **unicast** message queue group, each message is sent to only one of the message queues in the group. In a **multicast** message queue group, a message can be sent to more than one message queue in the group and, thus, can be received by more than one process.

There are several different unicast and multicast policies (defined below) that an implementation may support. The Equal Load Distribution is mandatory.

In the unicast and multicast policies given below, a **local member** is a message queue that is opened by a process residing on the same node as the sending process. In contrast, a **remote member** is a message queue that is opened by a process not residing on the same node as the sending process.

- **Equal Load Distribution (unicast)**

  The message is sent to a single member. The members are addressed one-by-one in a round-robin fashion. If an error occurs when sending to a member, it is implementation-dependent whether the error is returned immediately or whether the Message Service chooses the next member in turn and for how many members this is repeated. This policy is mandatory.

- **Local Equal Load Distribution (unicast)**

  The message is sent to a single member. The local member message queues are addressed one-by-one in a round-robin fashion. If there is no local member, the behavior is the same as for the equal load distribution policy. If an error occurs when sending to a member, it is implementation-dependent whether the error is immediately returned, or the Message Service chooses the next member in turn and for how many members this is repeated.

- **Local Best Queue (unicast)**

  The message is sent to a single member. The local member message queue with the largest amount of available space is selected. If several members fulfill this condition, they are addressed one-by-one in a round-robin fashion. If there is no local member, a remote member, or a member that is not opened by any process, is selected according to the equal load distribution policy. If an error occurs when sending to a member, it is implementation-dependent whether the error is returned immediately, or the Message Service chooses the next member in turn and for how many members this is repeated.

- **Broadcast (multicast)**

  The message is sent to all members of the message queue group that have sufficient space to hold the message.

3.1.4 Properties of Message Queues

3.1.4.1 Non-persistent and Persistent Message Queues

A message queue can be defined as **non-persistent**, meaning that the Message Service removes the message queue automatically from the cluster-wide name space, if it is not opened by a process for a configurable amount of time, called the

retention time. The retention time starts each time when the corresponding message queue is closed.

A **persistent** message queue is like a non-persistent message queue with an infinite retention time. A persistent message queue can only be removed by using the *saMsgQueueUnlink()* call.

If no process has the message queue opened when *saMsgQueueUnlink()* is called, the Message Service removes the message queue immediately, even if it is persistent; otherwise, the Message Service removes the message queue when it is closed.

3.1.4.2 Message Preservation Property of a Queue

If a persistent message queue or a message queue with non-zero retention time is closed, the Message Service must preserve all messages in the message queue that have not yet been consumed.

Example: In node switch-over situations, the Message Service must preserve messages in a queue, which have not yet been consumed. For example, assume a service unit, (for details, refer to [3]) containing a component *cx*, which retrieves messages from a message queue *Q* for the service assigned to this service unit. Assume further that this service switches over to another service unit, located on another node and containing the component *cy* that works as standby for *cx's* service. If *cy* reopens the queue *Q,* and it does not specify that it wants existing messages to be deleted, the Message Service must preserve the messages that were not retrieved by *cx* when *cx* closed *Q* as well as the messages that arrived at *Q* after *cx* closed it and before *cy* reopened it.

If message queues are implemented as node-local resources, it is not required that the Message Service preserve messages in case of node failures.

3.1.5 Associating Processes with Message Queues

Messages are sent and received by processes. The Message Service is a cooperative model where any process may write to any message queue or message queue group.

A process can retrieve (receive) messages from a message queue. For this purpose, a process can open a message queue, obtain a handle to it, and receive messages from it. While a process has a message queue open, the message queue cannot be opened again by the same process or by any other process.

If a process terminates abnormally, the Message Service automatically closes all of its open message queues.

3.1.6 Message Delivery Properties

- **Priority** - When a process receives a message from a message queue via an invocation of *saMsgMessageGet(),* it receives messages in a higher priority area before messages in a lower priority area. It receives messages of the same priority from a given process in the order in which that process sent them. It might not receive messages of different priorities from a given process in the same order in which that process sent them.
- **Integrity of messages** - The Message Service guarantees that messages sent by a process to a message queue are neither altered nor duplicated. Only complete messages are stored in a message queue.
- **At-most-once delivery** - The Message Service guarantees that a message sent to a message queue is delivered at most once to that message queue. For a message sent to a unicast message queue group, the message is delivered at most once to one of the member message queues. For a message sent to a multicast message queue group, the message is delivered at most once to each of the member message queues.
- **Delivery guarantees** - If a process sends a message to a message queue or to a unicast message queue group, and there is no space in the destination message queue for the entire message, the error code SA_AIS_ERR_QUEUE_FULL is returned, given that the process requested an acknowledgement for its send operation. It is expected that an implementation, for correctly constructed sending calls, returns errors other than SA_AIS_ERR_QUEUE_FULL only under exceptional and extremely rare conditions. For instance, it is not acceptable to drop packets due to a network that is momentarily congested. Therefore, the Message Service does not define a return value, such as communication error, for the sending API calls. Instead, it uses only a specific SA_AIS_ERR_TIMEOUT. If a process sends a message to a multicast message queue group, and it requests an acknowledgment, the sending call returns success if the message can be sent successfully to at least one member of the message queue group.
- **Acknowledgment** - A process sending messages can request the Message Service to notify the process whether the sending was successful. A process can ask only for an acknowledgment that the message has been stored in the destination message queue (refer to send operations in Section 3.7.1), or that the reply to a sending process has been received by the sending process (refer to send/reply operations in Section 3.8).
- **Persistence of messages** - Messages are kept in message queues. A message never expires in a message queue. When a message is retrieved successfully from a message queue through an invocation of *saMsgMessageGet(),* the message is removed from the message queue. The physical representation of a message queue may reside on disk, on a cluster

file system, on global shared memory, on shared memory of each node with or without replication, etc. However, the choice of persistence can have negative effects on the performance of the Message Service. Therefore, this specification does not require that the physical representation of a message queue be durably stored to survive node failures or shutting down the entire cluster.

3.2 Include File and Library Name

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

```
#include <saMsg.h>
```

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

libSaMsg.so

3.3 Type Definitions

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

3.3.1 Handles

3.3.1.1 SaMsgHandleT

```
typedef SaUint64T SaMsgHandleT;
```

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

3.3.1.2 SaMsgQueueHandleT

```
typedef SaUint64T SaMsgQueueHandleT;
```

The type of a handle to a message queue.

3.3.2 SaMsgSenderIdT

```
typedef SaUint64T SaMsgSenderIdT;
```

The type used internally by the Message Service to identify the thread that called *saMsgMessageSendReceive();* it must not be changed by the invoking thread.

3.3.3 SaMsgCallbacksT

The *SaMsgCallbacksT* structure is defined as follows:

```
typedef struct {
    SaMsgQueueOpenCallbackT saMsgQueueOpenCallback;
    SaMsgQueueGroupTrackCallbackT saMsgQueueGroupTrackCallback;
    SaMsgMessageDeliveredCallbackT saMsgMessageDeliveredCallback;
    SaMsgMessageReceivedCallbackT saMsgMessageReceivedCallback;
} SaMsgCallbacksT;
```

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

3.3.4 SaMsgAckFlagsT

The *SaMsgAckFlagsT* type is used in the *saMsgMessageSendAsync()* and *saMsgMessageReplyAsync()* calls. A parameter of the type *SaMsgAckFlagsT* indicates the kind of the required acknowledgment and can either be set to zero or SA_MSG_MESSAGE_DELIVERED_ACK:

```
#define SA_MSG_MESSAGE_DELIVERED_ACK 0x1
typedef SaUint32T SaMsgAckFlagsT;
```

SA_MSG_MESSAGE_DELIVERED_ACK - This flag indicates that the caller requires an acknowledgment to confirm whether the message can be stored in the destination message queue or reply buffer. If there is no space for the entire message in the destination message queue or reply buffer, the error SA_AIS_ERR_QUEUE_FULL is returned in case of a message queue, and SA_AIS_ERR_NO_SPACE is returned in case of a reply buffer.
If SA_MSG_MESSAGE_DELIVERED_ACK is not set, the caller does not require an acknowledgment.

3.3.5 Message Queue Creation Flags and Creation Attributes

This section defines the creation flags and the creation attributes of a message queue used in the *saMsgQueueOpen()* or *saMsgQueueOpenAsync()* calls.

3.3.5.1 SaMsgQueueCreationFlagsT

#define SA_MSG_QUEUE_PERSISTENT 0x1

typedef SaUint32T SaMsgQueueCreationFlagsT;

SA_MSG_QUEUE_PERSISTENT - If this flag is set, the message queue is persistent, that is, it can be removed only by an explicit call to *saMsgQueueUnlink*(). If this flag is not set, the message queue is non-persistent, that is, the Message Service removes the message queue automatically if it is not opened by a process for a *retentionTime* amount of time after its closure.

3.3.5.2 SaMsgQueueCreationAttributesT

```
typedef struct {
    SaMsgQueueCreationFlagsT creationFlags;
    SaSizeT size[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];
    SaTimeT retentionTime;
} SaMsgQueueCreationAttributesT;
```

The fields of the *SaMsgQueueCreationAttributesT* structure have the following interpretation:

- *creationFlags* - Zero or SA_MSG_QUEUE_PERSISTENT. Refer also to *retentionTime* below.
- *size* - The size in bytes of the priority area of the message queue to contain messages with the specified priority. For the meaning of SA_MSG_MESSAGE_LOWEST_PRIORITY, refer to Section 3.3.7 on page 24.
- *retentionTime* - The time duration after a process closes the message queue until the Message Service removes the message queue. This parameter applies only to non-persistent message queues.

3.3.6 SaMsgQueueOpenFlagsT

The following values specify the open attributes used in the *saMsgQueueOpen()* or *saMsgQueueOpenAsync()* calls.

#define SA_MSG_QUEUE_CREATE 0x1

#define SA_MSG_QUEUE_RECEIVE_CALLBACK 0x2

#define SA_MSG_QUEUE_EMPTY 0x4

typedef SaUint32T SaMsgQueueOpenFlagsT;

A value of the *SaMsgQueueOpenFlagsT* type is zero or the bitwise OR of one or more of the flags in the following list:

- SA_MSG_QUEUE_CREATE - This flag requests to create a message queue if the message queue does not already exist.
- SA_MSG_QUEUE_RECEIVE_CALLBACK - This flag requests the Message Service to notify the process about the arrival of messages via the *saMsgMessageReceivedCallback()* call. The message can be retrieved by invoking *saMsgMessageGet()*. If this flag is not set, the callback function for the arrival of messages is not called, but the user can still use the *saMsgMessageGet()* call to receive messages.
- SA_MSG_QUEUE_EMPTY - This flag requests the Message Service to delete existing messages when opening a message queue. If it is not set, the Message Service must preserve existing messages when opening the message queue. However, in a fail-over situation the Message Service cannot guarantee that messages will be preserved.

3.3.7 Message Priority

Messages can have a priority between SA_MSG_MESSAGE_LOWEST_PRIORITY and SA_MSG_MESSAGE_HIGHEST_PRIORITY.

#define SA_MSG_MESSAGE_HIGHEST_PRIORITY 0

#define SA_MSG_MESSAGE_LOWEST_PRIORITY 3

3.3.8 Message Queue Usage and Status

This section defines the structures *SaMsgQueueUsageT* and *SaMsgQueueStatusT* that are used to obtain information about a message queue.

3.3.8.1 SaMsgQueueUsageT

```
typedef struct {
    SaSizeT queueSize;
    SaSizeT queueUsed;
    SaUint32T numberOfMessages;
} SaMsgQueueUsageT;
```

The fields of the *SaMsgQueueUsageT* structure are valid for a given priority area of a message queue. They have the following interpretation:

- *queueSize* - The size in bytes of the priority area to hold messages of a particular priority.
- *queueUsed* - The current number of bytes of the priority area of the message queue occupied by messages of a particular priority.
- *numberOfMessages* - The current number of messages of a particular priority in the message queue.

3.3.8.2 SaMsgQueueStatusT

```
typedef struct {
    SaMsgQueueCreationFlagsT creationFlags;
    SaTimeT retentionTime;
    SaTimeT closeTime;
    SaMsgQueueUsageT saMsgQueueUsage
      [SA_MSG_MESSAGE_LOWEST_PRIORITY+1];
} SaMsgQueueStatusT;
```

The fields of the *SaMsgQueueStatusT* structure have the following interpretation:

- *creationFlags* - Zero or SA_MSG_QUEUE_PERSISTENT, which was defined in Section 3.3.5 on page 22.
- *retentionTime* - The time duration after a process closed the message queue until the Message Service removes it. This field applies only to non-persistent message queues.
- *closeTime* - The absolute time when the message queue was last closed. If the message queue is currently open for a process, this field contains zero.
- *saMsgQueueUsage* - An array containing the message queue usage data for each priority area of the message queue.

3.3.9 SaMsgQueueGroupPolicyT

typedef enum {

SA_MSG_QUEUE_GROUP_ROUND_ROBIN = 1,

SA_MSG_QUEUE_GROUP_LOCAL_ROUND_ROBIN = 2,

SA_MSG_QUEUE_GROUP_LOCAL_BEST_QUEUE = 3,

SA_MSG_QUEUE_GROUP_BROADCAST = 4

} SaMsgQueueGroupPolicyT;

The only mandatory message queue group policy in this version is SA_MSG_QUEUE_GROUP_ROUND_ROBIN. For the description of message queue group policies, refer to Section 3.1.3 on page 17.

3.3.10 Types for Tracking Message Queue Group Changes

This section defines the types needed for tracking message queue group changes.

3.3.10.1 SaMsgQueueGroupChangesT

typedef enum {

SA_MSG_QUEUE_GROUP_NO_CHANGE = 1,

SA_MSG_QUEUE_GROUP_ADDED = 2,

SA_MSG_QUEUE_GROUP_REMOVED = 3,

SA_MSG_QUEUE_GROUP_STATE_CHANGED = 4

} SaMsgQueueGroupChangesT;

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

- SA_MSG_QUEUE_GROUP_NO_CHANGE - This value is used when the *trackFlags* parameter of the *saMsgQueueGroupTrack*() function, defined in Section 3.6.5, is either
  - SA_TRACK_CURRENT or
  - SA_TRACK_CHANGES and
    - the message queue was already a member of the message queue group in the previous *saMsgQueueGroupTrackCallback()* callback call,
    - and it has not been removed from the message queue group.
- SA_MSG_QUEUE_GROUP_ADDED - The message queue has been added to the message queue group since the last callback.

- SA_MSG_QUEUE_GROUP_REMOVED - The message queue has been removed from the message queue group since the last callback.
- SA_MSG_QUEUE_GROUP_STATE_CHANGED - This value is reserved for future use.

3.3.10.2 SaMsgQueueGroupMemberT

```
typedef struct {
    SaNameT queueName;
} SaMsgQueueGroupMemberT;
```

The field *queueName* is the name of a message queue in the message queue group.

3.3.10.3 SaMsgQueueGroupNotificationT

```
typedef struct {
    SaMsgQueueGroupMemberT member;
    SaMsgQueueGroupChangesT change;
} SaMsgQueueGroupNotificationT;
```

The fields of the *SaMsgQueueGroupNotificationT* structure have the following interpretation:

- *member* - Information about a message queue group member.
- *change* - The type of change since the last callback.

3.3.10.4 SaMsgQueueGroupNotificationBufferT

```
typedef struct {
    SaUint32T numberOfItems;
    SaMsgQueueGroupNotificationT *notification;
    SaMsgQueueGroupPolicyT queueGroupPolicy,
} SaMsgQueueGroupNotificationBufferT;
```

The fields of the *SaMsgQueueGroupNotificationBufferT* structure have the following interpretation:

- *numberOfItems* - Number of elements of type *SaMsgQueueGroupNotificationT* in the notification buffer.
- *notification* - Start address of the notification buffer.
- *queueGroupPolicy* - The load distribution policy of the message queue group.

3.3.11 SaMsgMessageT

This structure describes a message to be used when sending and receiving messages.

```
typedef struct {
    SaUint32T type;
    SaUint32T version;
    SaSizeT size;
    SaNameT *senderName;
    void *data;
    SaUint8T priority;
} SaMsgMessageT;
```

The fields of the *SaMsgMessageT* structure have the following interpretation:

- *type* - Message type that is specified by a process when it sends a message.
- *version* - Version of the message, used to distinguish different versions of messages with the same message type. It is the responsibility of the application program to set this field and to ensure that the application program can handle messages with appropriate different versions.
- *size* - Size of the message data in bytes that is set by a process when it sends a message, and by the Message Service when it receives a message.
- *senderName* - This field identifies the sender of the message. It can be provided by a process sending the message. If the process sending the message is part of a component under the control of the Availability Management Framework, this field should contain the name of that component (in future, it is expected that in such cases it shall be mandatory to pass the LDAP DN of a component); otherwise, any octet string (including zeros) may be used as the sender name. If the sending process does not provide the sender name, but a receiving process expects it, the Message Service sets *senderName->length* to zero when the message is retrieved.
- *data* - A pointer to an area containing the message data. The message data is provided by a process when it sends a message. The Message Service passes the message data to a process when the process retrieves the message.
- *priority* - Priority of the message. This field is set by a process when it sends a message. For the possible values of this field, refer to Section 3.3.7 on page 24.

3.3.12 saMsgMessageCapacityStatusT

The following enum describes the various states of a message queue or a message queue group that need to be notified to a system administrator in form of a **state change** notification. For details on notifications, refer to Chapter 4 and [2].

For semantic clarity and flexibility regarding when such notifications are generated, the concept of a critical capacity per priority area within a message queue is introduced.

The **critical capacity** of a priority area is either its size or an implementation-dependent value pair which defines the low and high threshold values to use as definition of when critical capacity is reached; the high value defines entry into critical capacity status and the lower value when to leave critical capacity status. These values may be lower than the priority area size.

In future specifications, the critical capacity of a message queue is intended to be defined through Message Service configuration parameters.

typedef enum {

SA_MSG_QUEUE_CAPACITY_REACHED = 1,

SA_MSG_QUEUE_CAPACITY_AVAILABLE = 2,

SA_MSG_QUEUE_GROUP_CAPACITY_REACHED = 3,

SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE = 4

} SaMsgMessageCapacityStatusT;

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

- SA_MSG_QUEUE_CAPACITY_REACHED - All priority areas of a message queue are at critical capacity, and this condition is potentially affecting the capability of the message queue to accept new messages in any of its priority areas.
- SA_MSG_QUEUE_CAPACITY_AVAILABLE - At least one priority area in the message queue is no longer filled up to its critical capacity and is available to accept new messages after having recovered from a preceding SA_MSG_QUEUE_CAPACITY_REACHED condition.
- SA_MSG_QUEUE_GROUP_CAPACITY_REACHED - All priority areas of all the message queues within a message queue group are filled up to their critical capacities, and this is potentially affecting the capability of the message queue group to accept new messages.
- SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE - At least one data area in one message queue within the queue group is no longer filled up to its critical

capacity and is available to accept new messages after having recovered from a previous SA_MSG_QUEUE_GROUP_CAPACITY_REACHED condition.

3.3.13 saMsgStateT

The following enum holds all the message queue state types. Currently, there is only one such state:

```
typedef enum {
    SA_MSG_DEST_CAPACITY_STATUS = 1
} SaMsgStateT;
```

3.4 Library Life Cycle

General remark: If a library call of the Message Service does not complete due to a crash of a process, or if a timeout indicated by SA_AIS_ERR_TIMEOUT is returned to the process, it is unspecified whether the corresponding function succeeded or whether it did not.

3.4.1 saMsgInitialize()

Prototype

```
SaAisErrorT saMsgInitialize(
    SaMsgHandleT *msgHandle,
    const SaMsgCallbacksT *msgCallbacks,
    SaVersionT *version
);
```

Parameters

msgHandle - [out] A pointer to the handle designating this particular initialization of the Message Service.

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

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

As an output parameter, the version actually supported by the Message Service is delivered.

Description

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

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

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

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

if (implementation supports the required *releaseCode*)

releaseCode = required *releaseCode*

else *{*

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

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

else

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

}

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it 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 Message Service library or the provider of the service is out of memory and cannot provide the service.

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

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

See Also

saMsgSelectionObjectGet(), saMsgDispatch(), saMsgFinalize()

3.4.2 saMsgSelectionObjectGet()

Prototype

```
SaAisErrorT saMsgSelectionObjectGet(
    SaMsgHandleT msgHandle,
    SaSelectionObjectT *selectionObject
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

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

Description

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

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

The *selectionObject* returned by *saMsgSelectionObjectGet()* is valid until *saMsgFinalize()* is invoked on the same handle *msgHandle*.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

SA_AIS_ERR_NO_RESOURCES - There are not enough resources (other than memory).

See Also

saMsgInitialize(), saMsgDispatch(), saMsgFinalize()

3.4.3 saMsgDispatch()

Prototype

```
SaAisErrorT saMsgDispatch(
        SaMsgHandleT msgHandle,
        SaDispatchFlagsT dispatchFlags
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

dispatchFlags - [in] Flags that specify the callback execution behavior of the *saMsgDispatch()* function, which have the values SA_DISPATCH_ONE, SA_DISPATCH_ALL, or SA_DISPATCH_BLOCKING, as defined in the SA Forum Overview document.

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it 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 *msgHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INVALID_PARAM - The *dispatchFlags* parameter is invalid.

See Also

saMsgInitialize(), saMsgSelectionObjectGet(), saMsgFinalize()

3.4.4 saMsgFinalize()

Prototype

```
SaAisErrorT saMsgFinalize(
        SaMsgHandleT msgHandle
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

Description

The *saMsgFinalize()* function closes the association, represented by the *msgHandle* parameter, between the invoking process and the Message Service. The process must have invoked *saMsgInitialize()* before it invokes this function. A process must invoke this function once for each handle it acquired by invoking *saMsgInitialize()*.

If the s*aMsgFinalize()* function returns successfully, the *saMsgFinalize()* function releases all resources acquired when *saMsgInitialize()* was called. Moreover, it closes all message queues that are open for the particular handle. Furthermore, it stops any tracking associated with the particular handle and cancels all pending callbacks related to the particular handle. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

After *saMsgFinalize()* is called, the selection object is no longer valid.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it 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 *msgHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

See Also

saMsgInitialize(), saMsgSelectionObjectGet(), saMsgDispatch()

3.5 Message Queue Operations

In the following description, when it is said that a process is receiving from the *destination* message queue, it also includes the case of a process receiving from a message queue that is a member of a *destination* message queue group.

3.5.1 saMsgQueueOpen() and saMsgQueueOpenAsync()

The *saMsgQueueOpen()* and *saMsgQueueOpenAsync()* functions create and open a new message queue, or open an existing message queue. The *saMsgQueueOpen()* function is a synchronous blocking operation that returns a new message queue handle. The *saMsgQueueOpenAsync()* function is an asynchronous operation; the corresponding s*aMsgQueueOpenCallback()* returns the new message queue handle to the invoking process.

When opening a message queue, a process can specify how it will receive messages:

- **Arrival notified by a callback** - The process can wait for an indication of a message arrival using the operating system handle associated with the callback. After being notified, the process can invoke *saMsgDispatch(),* which, in turn, calls *saMsgMessageReceivedCallback().* In the callback, or after its completion, the process can invoke *saMsgMessageGet()* to receive the message.

- **Through a blocking call** - The process can call *saMsgMessageGet()* directly without using the selection object. The call will complete successfully if a message can be retrieved from the message queue within a specified time limit.

Prototype

```
SaAisErrorT saMsgQueueOpen(
        SaMsgHandleT msgHandle,
        const SaNameT *queueName,
        const SaMsgQueueCreationAttributesT *creationAttributes,
        SaMsgQueueOpenFlagsT openFlags,
        SaTimeT timeout,
        SaMsgQueueHandleT *queueHandle
);

SaAisErrorT saMsgQueueOpenAsync(
        SaMsgHandleT msgHandle,
        SaInvocationT invocation,
        const SaNameT *queueName,
        const SaMsgQueueCreationAttributesT *creationAttributes,
        SaMsgQueueOpenFlagsT openFlags
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

invocation - [in] The invoking process supplies the *invocation* parameter and the Message Service uses this *invocation* when it invokes the corresponding *saMsgQueueOpenCallback()* function to enable the invoking process to associate the callback with the appropriate invocation of *saMsgQueueOpenAsync()*.

queueName - [in] A pointer to the name of the message queue to be opened.

creationAttributes - [in] A pointer to the creation attributes of a message queue.

If the intent is only to open an existing message queue, *creationAttributes* must be set to NULL and the SA_MSG_QUEUE_CREATE flag in *openFlags* must not be set. If the intent is to open and create a message queue if it does not exist, *creationAttributes* must contain the attributes for the message queue and the SA_MSG_QUEUE_CREATE flag in *openFlags* must be set. If the message queue already exists, it is not re-created, and the call only succeeds if the creation attributes match the ones used at creation time, excluding *creationAttributes->retentionTime,* which is ignored, as it may be independently modified by invoking the *saMsgQueueRetentionTimeSet()* function.

- *creationFlags* - The *creationFlags* field of the *creationAttributes* parameter must be set to zero or to SA_MSG_QUEUE_PERSISTENT, which was defined in Section 3.3.5 on page 22.
- *size* - The size in bytes of all priority areas of the message queue.
- *retentionTime* - The time duration after a process closed the message queue and until it is removed by the Message Service. The *retentionTime* applies only to non-persistent message queues.

openFlags - [in] This parameter is evaluated at open time, and it is the bitwise OR of the SA_MSG_QUEUE_CREATE, SA_MSG_QUEUE_RECEIVE_CALLBACK, and SA_MSG_QUEUE_EMPTY flags, defined in Section 3.3.6 on page 23.

timeout - [in] The *saMsgQueueOpen()* invocation is considered to have failed if it does not complete within the duration specified.

queueHandle - [out] A pointer to the handle assigned by the Message Service to the message queue. The invoking process must allocate space for the handle before it invokes the *saMsgQueueOpen()* function.

Description

The *saMsgQueueOpen()* and *saMsgQueueOpenAsync()* functions open a message queue. If the message queue does not exist and the SA_MSG_QUEUE_CREATE flag is set in the *openFlags* parameter, the message queue is created first.

After completion of the invocation of *saMsgQueueOpen(),* which is a blocking call, the Message Service returns a message queue handle to the message queue in the *queueHandle* parameter.

For *saMsgQueueOpenAsync(),* the Message Service returns a message queue handle when it invokes the s*aMsgQueueOpenCallback()* function, which must have been supplied when the process invoked the *saMsgInitialize()* call. The process invoking *saMsgQueueOpenAsync()* sets the *invocation* parameter and the Message Service uses it in the corresponding callback call.

Both for the *saMsgQueueOpen()* and *saMsgQueueOpenAsync()* functions, if the message queue open flag SA_MSG_QUEUE_RECEIVE_CALLBACK is specified, the *saMsgMessageReceivedCallback()* callback function must have been supplied when invoking *saMsgInitialize()* previously.

The open operation is needed to receive messages from the message queue.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INIT - One or more callback functions were not supplied in the previous initialization with *saMsgInitialize().* These callback functions can be either *saMsgMessageReceivedCallback(),* if the message queue open flag SA_MSG_QUEUE_RECEIVE_CALLBACK is specified or s*aMsgQueueOpenCallback().* The latter case only applies to *saMsgQueueOpenAsync()*.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this value is returned if one of the cases below apply:

- The SA_MSG_QUEUE_CREATE flag is not set, and *creationAttributes* is not NULL.
- The SA_MSG_QUEUE_CREATE flag is set, and *creationAttributes* is NULL.
- The SA_MSG_QUEUE_CREATE flag is set in *openFlags*, and *queueName* is not a DN or the type of its first RDN is not *safMq*.

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

SA_AIS_ERR_NO_RESOURCES - There are not enough resources (other than memory).

SA_AIS_ERR_NOT_EXIST - The SA_MSG_QUEUE_CREATE flag is not set, and the message queue, designated by *queueName,* does not exist.

SA_AIS_ERR_EXIST - The message queue, designated by *queueName,* already exists, and one or both of the values *creationAttributes->creationFlags* or *creationAttributes->size* are different from the corresponding values used at creation time.

SA_AIS_ERR_BUSY - The message queue, designated by *queueName,* is already open.

SA_AIS_ERR_BAD_FLAGS - The *creationFlags* field of the *creationAttributes* parameter or the *openFlags* parameter is invalid.

See Also

saMsgQueueClose(), SaMsgQueueOpenCallbackT, SaMsgMessageReceivedCallbackT, saMsgQueueRetentionTimeSet()

3.5.2 SaMsgQueueOpenCallbackT

Prototype

```
typedef void(*SaMsgQueueOpenCallbackT)(
        SaInvocationT invocation,
        SaMsgQueueHandleT queueHandle,
        SaAisErrorT error
);
```

Parameters

invocation - [in] A designator that associates this invocation to a previous call to the *saMsgQueueOpenAsync()* function.

queueHandle - [in] The handle to the opened message queue.

error - [in] The *error* parameter specifies whether the corresponding invocation of *saMsgQueueOpenAsync()* succeeded or not. The possible values of the *error* parameter are:

- SA_AIS_OK - The open 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 *msgHandle* in the corresponding invocation of the *saMsgQueueOpenAsync()* function is invalid, since it is corrupted, uninitialized, or has already been finalized.
- SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly in the corresponding invocation of the *saMsgQueueOpenAsync()* function. In particular, this value is returned if one of the cases below apply:
  - The SA_MSG_QUEUE_CREATE flag is not set, and *creationAttributes* is not NULL.
  - The SA_MSG_QUEUE_CREATE flag is set, and *creationAttributes* is NULL.
  - The SA_MSG_QUEUE_CREATE flag is set in *openFlags*, and *queueName* is not a DN or the type of its first RDN is not *safMq*.
- SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the provider of the service is out of memory and cannot provide the service.
- SA_AIS_ERR_NO_RESOURCES - There are not enough resources (other than memory).
- SA_AIS_ERR_NOT_EXIST - In the corresponding invocation of the *saMsgQueueOpenAsync()* function, the SA_MSG_QUEUE_CREATE flag is not set, and the message queue, designated by *queueName,* does not exist.
- SA_AIS_ERR_EXIST - The message queue, designated by *queueName* in the corresponding invocation of the *saMsgQueueOpenAsync()* function, already exists, and one or both of the values *creationAttributes->creationFlags* or *creationAttributes->size* are different from the corresponding values used at creation time.
- SA_AIS_ERR_BUSY - The message queue, designated by *queueName* in the corresponding invocation of the *saMsgQueueOpenAsync()* function, is already open.
- SA_AIS_ERR_BAD_FLAGS - In the corresponding invocation of the *saMsgQueueOpenAsync()* function, the *creationFlags* field of the *creationAttributes* parameter or the *openFlags* parameter is invalid.

Description

The Message Service invokes this callback function when the operation requested by the invocation of *saMsgQueueOpenAsync()* completes. This callback is invoked in the context of a thread issuing an *saMsgDispatch()* call on the handle *msgHandle,* which was specified in the *saMsgQueueOpenAsync()* call. The reference to the

opened/created message queue is returned in *queueHandle* only if *error* is SA_AIS_OK*;* If the call is not successful, an error is returned in the error parameter.

Return Values

None

See Also

saMsgQueueOpenAsync(), saMsgQueueClose(), saMsgDispatch()

3.5.3 saMsgQueueClose()

Prototype

```
SaAisErrorT saMsgQueueClose(
        SaMsgQueueHandleT queueHandle
);
```

Parameters

queueHandle - [in] The handle to the message queue to be closed.

Description

This API function closes the message queue specified by *queueHandle.* After this call, the handle *queueHandle* is no longer valid.

The Message Service will immediately delete the message queue in the following cases:

- *saMsgQueueUnlink()* has been invoked for the message queue while it was open.
- The message queue is non-persistent, and its retention time is zero.

If the message queue is non-persistent with a retention time greater than zero, the retention time starts when the *saMsgQueueClose()* call completes successfully. If the message queue is not opened again before this time elapses, the Message Service deletes the message queue.

The deletion of a message queue frees all resources allocated by the Message Service for it.

When a message queue is deleted, it is also deleted from all message queue groups it is a member of.

If a process terminates, the Message Service implicitly closes all message queues that are open for this process.

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

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it 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 *queueHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained via the *saMsgQueueOpen()* or *saMsgQueueOpenCallback()* functions, or the corresponding message queue has already been closed.
- The handle *msgHandle* that was passed to the functions *saMsgQueueOpen*() or *saMsgQueueOpenAsync*() has already been finalized.

See Also

saMsgQueueOpen(), saMsgQueueOpenAsync(), SaMsgQueueOpenCallbackT

3.5.4 saMsgQueueStatusGet()

Prototype

SaAisErrorT saMsgQueueStatusGet(
SaMsgHandleT msgHandle,
*const SaNameT *queueName,*
*SaMsgQueueStatusT *queueStatus*
);

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueName - [in] A pointer to the name of the message queue whose communication status is to be retrieved.

queueStatus - [out] A pointer to the structure, allocated by the invoking process, that contains status information on the message queue identified by *queueName* supplied by the Message Service.

Description

This function retrieves information about the status of a message queue.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

SA_AIS_ERR_NOT_EXIST - The message queue, designated by *queueName,* cannot be found.

See Also

-

3.5.5 saMsgQueueRetentionTimeSet()

Prototype

SaAisErrorT saMsgQueueRetentionTimeSet(

SaMsgQueueHandleT queueHandle,

*SaTimeT *retentionTime*

);

Parameters

queueHandle - [in] The handle to the message queue for which the retention time is set.

retentionTime - [in] The value of the retention time to be set for the message queue designated by *queueHandle*.

Description

The *saMsgQueueRetentionTimeSet()* function sets the retention time of the message queue, designated by *queueHandle*, to *retentionTime*. When the message queue is closed and not reopened by any process within the duration given by *retentionTime*, the Message Service deletes the message queue. The retention time can only be set for non-persistent message queues.

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

- It is corrupted, was not obtained via the *saMsgQueueOpen()* or *saMsgQueueOpenCallback()* functions, or the corresponding message queue has already been closed.
- The handle *msgHandle* that was passed to the functions *saMsgQueueOpen(),* or *saMsgQueueOpenAsync()* has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_BAD_OPERATION - The retention time of the message queue designated by *queueHandle* cannot be changed as the message queue has been unlinked or the message queue is persistent.

See Also

saMsgQueueOpen(), saMsgQueueOpenAsync(), saMsgQueueClose(), saMsgQueueOpenCallbackT, saMsgQueueUnlink()

3.5.6 saMsgQueueUnlink()

Prototype

```
SaAisErrorT saMsgQueueUnlink(
        SaMsgHandleT msgHandle,
        const SaNameT *queueName
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueName - [in] A pointer to the name of the message queue to be unlinked.

Description

This function deletes an existing message queue, identified by *queueName*, from the cluster.

After completion of the invocation:

- The name *queueName* is no longer valid, that is, any invocation of a function of the Message Service API that uses the message queue name returns an error, unless a message queue is re-created with this name. The message queue is re-created by specifying the same name of the message queue to be unlinked in an *saMsgQueueOpen()* or an *saMsgQueueOpenAsync()* call with the SA_MSG_QUEUE_CREATE flag set. This way, a new instance of the message queue is created while the old instance of the message queue is possibly not yet finally deleted.
  Note that this is similar to the way POSIX treats files.
- If no process has the message queue open when *saMsgQueueUnlink()* is invoked, the message queue is immediately deleted, even if its creation attribute is SA_MSG_QUEUE_PERSISTENT.
- The process that has the message queue open can still continue to access it. Deletion of the message queue will occur when the message queue is closed.

The deletion of a message queue frees all resources allocated by the Message Service for it.

When *saMsgQueueUnlink()* has successfully completed, the message queue has been removed from all message queue groups it is a member of.

This API can be invoked by any process and the invoking process need not be the creator or opener of the message queue.

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

SA_AIS_ERR_NOT_EXIST - The message queue, designated by *queueName,* cannot be found.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saMsgQueueOpen(), saMsgQueueOpenAsync(), saMsgQueueClose()

3.6 Management of Message Queue Groups

3.6.1 saMsgQueueGroupCreate()

Prototype

```
SaAisErrorT saMsgQueueGroupCreate(
        SaMsgHandleT msgHandle,
        const SaNameT *queueGroupName,
        SaMsgQueueGroupPolicyT queueGroupPolicy
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName -[in] A pointer to the name of a message queue group to be created.

queueGroupPolicy - [in] The message queue group policy. Currently, only the SA_MSG_QUEUE_GROUP_ROUND_ROBIN policy is mandatory.

Description

This function creates a message queue group of a particular policy. The current version of the specification only requires round robin load distribution to be supported.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this value is returned if *queueGroupName* is not a DN, or the type of its first RDN is not *safMqg*.

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

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

SA_AIS_ERR_EXIST - The message queue group, identified by *queueGroupName,* exists already.

SA_AIS_ERR_NOT_SUPPORTED - The specified *queueGroupPolicy* is not supported by the implementation.

See Also

saMsgQueueGroupDelete(), saMsgQueueGroupInsert(), saMsgQueueGroupRemove()

3.6.2 saMsgQueueGroupInsert()

Prototype

```
SaAisErrorT saMsgQueueGroupInsert(
    SaMsgHandleT msgHandle,
    const SaNameT *queueGroupName,
    const SaNameT *queueName
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName - [in] A pointer to the name of the message queue group into which the message queue, indicated by *queueName,* is to be inserted.

queueName - [in] A pointer to the name of the message queue to be inserted into the message queue group *queueGroupName*.

Description

This function inserts a message queue into a message queue group.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_NOT_EXIST - The message queue, designated by *queueName,* or the message queue group, designated by *queueGroupName,* cannot be found.

SA_AIS_ERR_EXIST - The message queue, identified by *queueName,* is already a member of the message queue group, identified by *queueGroupName*.

See Also

saMsgQueueGroupRemove(), saMsgQueueGroupCreate(), saMsgQueueGroupDelete()

3.6.3 saMsgQueueGroupRemove()

Prototype

SaAisErrorT saMsgQueueGroupRemove(

SaMsgHandleT msgHandle,

*const SaNameT *queueGroupName,*

*const SaNameT *queueName*

);

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName - [in] A pointer to the name of the message queue group from which the message queue, indicated by *queueName,* is to be removed.

queueName - [in] A pointer to the name of the message queue to be removed from the message queue group *queueGroupName*.

Description

This function removes a message queue from a message queue group.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NOT_EXIST - This error is returned in the two cases that follow:

- The message queue, designated by *queueName,* or the message queue group, designated by *queueGroupName,* cannot be found.
- The message queue, designated by *queueName,* is not a member of the message queue group designated by *queueGroupName*.

See Also

saMsgQueueGroupInsert(), saMsgQueueGroupCreate(), saMsgQueueGroupDelete()

3.6.4 saMsgQueueGroupDelete()

Prototype

```
SaAisErrorT saMsgQueueGroupDelete(
        SaMsgHandleT msgHandle,
        const SaNameT *queueGroupName
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName - [in] A pointer to the name of a message queue group.

Description

An invocation of this function deletes a message queue group immediately. After this call, it is no longer possible to send messages to the message queue group.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NOT_EXIST - The message queue group, identified by *queueGroupName,* cannot be found.

See Also

s*aMsgQueueGroupCreate(), saMsgQueueGroupInsert(), saMsgQueueGroupRemove()*

3.6.5 saMsgQueueGroupTrack()

Prototype

```
SaAisErrorT saMsgQueueGroupTrack(
    SaMsgHandleT msgHandle,
    const SaNameT *queueGroupName,
    SaUint8T trackFlags,
    SaMsgQueueGroupNotificationBufferT *notificationBuffer
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName - [in] A pointer to the name of the message queue group for which tracking of membership is to start.

trackFlags - [in] The kind of tracking that is requested, which is the bitwise OR of one or more of the flags SA_TRACK_CURRENT, SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, defined in the SA Forum Overview document, which have the following interpretation here:

- SA_TRACK_CURRENT - If *notificationBuffer* is NULL, information about all members in the message queue group is returned by a single subsequent invocation of the *saMsgQueueGroupTrackCallback()* notification callback; otherwise, this information is returned in *notificationBuffer* when the *saMsgQueueGroupTrack()* call completes successfully.

- SA_TRACK_CHANGES - The notification callback is invoked each time a change occurs in the membership of the message queue group. The callback call provides an *SaMsgQueueGroupNotificationT* structure for all members (changed and not changed) of the message queue group.
- SA_TRACK_CHANGES_ONLY - The notification callback is invoked each time a change occurs in the membership of the message queue group. The callback call provides an *SaMsgQueueGroupNotificationT* structure only for members that have changed.

It is not permitted to set both SA_TRACK_CHANGES and SA_TRACK_CHANGES_ONLY in an invocation of this function.

notificationBuffer - [in/out] - A pointer to a buffer of type *SaMsgQueueGroupNotificationBufferT*. This parameter is ignored if SA_TRACK_CURRENT is not set in *trackFlags*; otherwise, if *notificationBuffer* is not NULL, the buffer will contain information about all members in the message queue group when *saMsgQueueGroupTrack()* returns. The meaning of the fields of the *SaMsgQueueGroupNotificationBufferT* buffer is:

- *numberOfItems* - [in/out] If *notification* is NULL, *numberOfItems* is ignored as input parameter; otherwise, it specifies that the buffer pointed to by *notification* provides memory for information about *numberOfItems* members in the message queue group.
  When *saMsgQueueGroupTrack()* returns with SA_AIS_OK or with SA_AIS_ERR_NO_SPACE, *numberOfItems* contains the number of members in the message queue group.
- *notification* - [in/out] If *notification* is NULL, memory for the message queue group information is allocated by the Message Service library. The caller is responsible for freeing the allocated memory by calling the *saMsgQueueGroupNotificationFree()* function.

Description

This function starts tracking changes in the membership of a message queue group, identified by *queueGroupName*. These changes are notified via the invocation of the s*aMsgQueueGroupTrackCallback()* callback function, which must have been supplied when the process invoked the *saMsgInitialize()* call.

An application may call *saMsgQueueGroupTrack()* repeatedly for the same values of *msgHandle* and *queueGroupName*, regardless of whether the call initiates a one-time status request or a series of callback notifications. If *saMsgQueueGroupTrack()* is called with *trackFlags* containing SA_TRACK_CHANGES_ONLY, while changes in the membership of a message queue group are currently being tracked with SA_TRACK_CHANGES for the same combination of *msgHandle* and

queueGroupName, the Message Service will invoke further notification callbacks according to SA_TRACK_CHANGES_ONLY. The same is true vice versa.
Once *saMsgQueueGroupTrack()* has been called with *trackFlags* containing either SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, notification callbacks can only be stopped by an invocation of the *saMsgQueueGroupTrackStop()* function.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it 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 *msgHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with *saMsgInitialize()* was incomplete, since the s*aMsgQueueGroupTrackCallback()* callback function is missing. This value is not returned if *trackFlags* is set to SA_TRACK_CURRENT, and the *notificationBuffer* is not NULL.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this applies if, in *notificationBuffer, notification* is not NULL, and *numberOfItems* is 0.

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

SA_AIS_ERR_NO_RESOURCES -The system is out of required resources (other than memory).

SA_AIS_ERR_NO_SPACE - The SA_TRACK_CURRENT flag is set, and the *notification* field in *notificationBuffer* is not NULL, but the *numberOfItems* field in *notificationBuffer* indicates that the provided buffer is too small to hold information about all members in the message queue group designated by *queueGroupName*.

SA_AIS_ERR_NOT_EXIST - The message queue group name, designated by *queueGroupName,* cannot be found.

SA_AIS_ERR_BAD_FLAGS - The *trackFlags* parameter is invalid.

See Also

saMsgQueueGroupTrackStop(), SaMsgQueueGroupTrackCallbackT, saMsgQueueGroupNotificationFree(), saMsgQueueStatusGet()

3.6.6 SaMsgQueueGroupTrackCallbackT

Prototype

```
typedef void (*SaMsgQueueGroupTrackCallbackT) (
    const SaNameT *queueGroupName,
    const SaMsgQueueGroupNotificationBufferT *notificationBuffer,
    SaUint32T numberOfMembers,
    SaAisErrorT error
);
```

Parameters

queueGroupName - [in] A pointer to the name of the message queue group.

notificationBuffer - [in] A pointer to a notification buffer, which contains the requested information about the members in the message queue group.

numberOfMembers - [in] The current number of members in the message queue group given by *queueGroupName.*

error - [in] This parameter indicates whether the Message Service was able to perform the operation. The parameter *error* has one of the 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 the *saMsgQueueGroupTrack()* call later.
- SA_AIS_ERR_BAD_HANDLE - The handle *msgHandle* in the corresponding invocation of the *saMsgQueueGroupTrack()* function is invalid, since it is corrupted, uninitialized, or has already been finalized.

- SA_AIS_ERR_INVALID_PARAM - In the corresponding invocation of the *saMsgQueueGroupTrack()* function, a parameter is not set correctly.
- SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the provider of the service is out of memory and cannot provide the service. The process that invoked *saMsgQueueGroupTrack()* might have missed one or more notifications.
- SA_AIS_ERR_NO_RESOURCES - Either the Message Service library or the provider of the service is out of resources (other than memory), and cannot provide the service. The process that invoked *saMsgQueueGroupTrack()* might have missed one or more notifications.
- SA_AIS_ERR_NOT_EXIST - The message queue group, designated by *queueGroupName,* no longer exists. The Message Service has stopped the tracking of the message queue group automatically.
- SA_AIS_ERR_BAD_FLAGS - The *trackFlags* parameter in the corresponding invocation of the *saMsgQueueGroupTrack()* function is invalid.

If the error returned is SA_AIS_ERR_NO_MEMORY or SA_AIS_ERR_NO_RESOURCES, the process that invoked *saMsgQueueGroupTrack()* should invoke *saMsgQueueGroupTrackStop().* It may then invoke *saMsgQueueGroupTrack()* again.

Description

This callback is invoked in the context of a thread issuing an *saMsgDispatch()* call on the handle *msgHandle,* which was specified when the process requested tracking of membership in a message queue group, or changes in the *SaMsgQueueGroupMemberT* structure of any member of the message queue group via the *saMsgQueueGroupTrack()* call. If successful, the *saMsgQueueGroupTrackCallback()* function returns information about the message queue group members in the *notificationBuffer* parameter. The kind of information returned depends on the setting of the *trackFlags* parameter of the *saMsgQueueGroupTrack()* function.

The value of the *numberOfItems* attribute in the *notificationBuffer* parameter might be greater than the value of the *numberOfMembers* parameter, because some message queues may no longer be members of the message queue group: If the SA_TRACK_CHANGES flag or the SA_TRACK_CHANGES_ONLY flag is set, the *notificationBuffer* might contain information about the current members of the message queue group and also about message queues that have recently left the message queue group.

If an error occurs, it is returned in the error parameter.

Return Values

None.

See Also

saMsgQueueGroupTrack(), saMsgQueueGroupTrackStop(), saMsgQueueStatusGet(), saMsgDispatch()

3.6.7 saMsgQueueGroupTrackStop()

Prototype

```
SaAisErrorT saMsgQueueGroupTrackStop(
        SaMsgHandleT msgHandle,
        const SaNameT *queueGroupName
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

queueGroupName - [in] A pointer to the name of the message queue group.

Description

This function requests the Message Service to stop tracking changes for the message queue group identified by *queueGroupName.*

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

SA_AIS_ERR_NO_RESOURCES -The system is out of required resources (other than memory).

SA_AIS_ERR_NOT_EXIST - This value is returned if one or both cases below occurred:

- The message queue group name, designated by *queueGroupName,* cannot be found.
- No track of changes in the membership in *queueGroupName w*as previously started via *saMsgQueueGroupTrack()* with track flags SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, and which is still in effect.

See Also

saMsgQueueGroupTrack(), SaMsgQueueGroupTrackCallbackT

3.6.8 saMsgQueueGroupNotificationFree()

Prototype

```
SaAisErrorT saMsgQueueGroupNotificationFree(
        SaMsgHandleT msgHandle,
        SaMsgQueueGroupNotificationT *notification
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

notification - [in] A pointer to the notification buffer that was allocated by the Message Service library in the *saMsgQueueGroupTrack()* function and is to be deallocated.

Description

This function frees the memory pointed to by *notification* and that was allocated by the Message Service library in a previous call to the *saMsgQueueGroupTrack()* function.

For details, refer to the description of the *notificationBuffer* parameter in the corresponding invocation of the *saMsgQueueGroupTrack()* function.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saMsgQueueGroupTrack()

3.7 Message Send and Receive Operations

3.7.1 saMsgMessageSend() and saMsgMessageSendAsync()

Prototype

```
SaAisErrorT saMsgMessageSend(
        SaMsgHandleT msgHandle,
        const SaNameT *destination,
        const SaMsgMessageT *message,
        SaTimeT timeout
);

SaAisErrorT saMsgMessageSendAsync(
        SaMsgHandleT msgHandle,
        SaInvocationT invocation,
        const SaNameT *destination,
        const SaMsgMessageT *message,
        SaMsgAckFlagsT ackFlags
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

invocation - [in] This parameter associates this invocation of *saMsgMessageSendAsync()* with a corresponding invocation of the *saMsgMessageDeliveredCallback()* function. This parameter is ignored if *ackFlags* is set to zero, meaning that the *saMsgMessageDeliveredCallback()* function is not called, and the caller is not informed whether an error occurred or whether it did not.

destination - [in] A pointer to the name of a message queue or message queue group the message, designated by *message,* is sent to.

message - [in] A pointer to the message structure specifying the message to be sent and consisting of a data field (a buffer provided by the process that contains the data to be sent), a *size* field that contains the size of the buffer, a *type* field that contains

the message type, a *version* field that is used to distinguish different versions of a message of the same type, a *priority* field that gives the priority of the message, and a *senderName* pointer. If *senderName* is not NULL, it points to an area containing the sender name, which is supplied by the caller; if *senderName* is NULL, the sending process does not provide its sender name, and a receiving process expecting a sender name will get
a *message->senderName->length* set to zero.

ackFlags - [in] The kind of the required acknowledgment. This field must be set to zero or to SA_MSG_MESSAGE_DELIVERED_ACK. In the latter case, the caller requires to be acknowledged whether the message can be stored in the *destination* message queue. If there is no space for the entire message in the destination message queue, the error SA_AIS_ERR_QUEUE_FULL is returned.

timeout - [in] The *saMsgMessageSend()* invocation is considered to have failed if it does not complete within the duration specified.

Description

The functions *saMsgMessageSend()* and *saMsgMessageSendAsync()* send the message, designated by *message,* to the message queue or message queue group, designated by *destination*.

The function *saMsgMessageSend()* waits synchronously (that is, it blocks) until the message is delivered to the destination message queue or message queue group, or an error occurs.
After the *saMsgMessageSend()* function returns, the invoking process may deallocate the memory for the data in the *message* buffer.

The function *saMsgMessageSendAsync()* returns as soon as possible, without waiting for delivery to the destination message queue. If the value of the *ackFlags* field is zero, the *saMsgMessageDeliveredCallback()* is not invoked, and the caller is not informed if an error occurs. If the value of the *ackFlags* field is set to SA_MSG_MESSAGE_DELIVERED_ACK, and this call returns SA_AIS_OK, *saMsgMessageDeliveredCallback()* is invoked to indicate whether the message was sent to the destination, or whether an error occurred. For this purpose, the user must have supplied the *saMsgMessageDeliveredCallback()* when invoking the *saMsgInitialize()* function.
If *saMsgMessageSendAsync()* returns successfully, and *ackFlags* is not set to zero, the sending process may deallocate the memory for the data in the *message* buffer either during an invocation of *saMsgMessageDeliveredCallback()* or after *saMsgMessageDeliveredCallback()* returns.
If *saMsgMessageSendAsync()* returns an error, or *ackFlags* is set to zero, meaning that *saMsgMessageDeliveredCallback()* will not be called, the process may deallo-

cate the memory for the data in the message buffer as soon as *saMsgMessageSendAsync()* returns.

Message delivery properties:

These properties apply to either a destination message queue or a message queue that is a member of a destination message queue group.

saMsgMessageSend():

- Message queue or a unicast message queue group - If the return value is SA_AIS_OK, the message has been delivered to exactly one destination message queue; otherwise, if the return value is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been delivered to any destination message queue.
- Multicast message queue group - If the return value is SA_AIS_OK, the message has been delivered to at least one member of the destination message queue group; otherwise, if the return value is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been delivered to any destination message queue.

saMsgMessageSendAsync():
If *saMsgMessageSendAsync()* returns SA_AIS_OK, and if the value of the *ackFlags* field is set to SA_MSG_MESSAGE_DELIVERED_ACK, the Message Service will invoke *saMsgMessageDeliveredCallback()*; otherwise, if the error code is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the Message Service will not invoke *saMsgMessageDeliveredCallback(),* and will not deliver the message to any destination message queue. Refer to the *saMsgMessageDeliveredCallback()* function for the message delivery properties.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred, or the timeout defined by the *timeout* parameter occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not. This error code applies only to *saMsgMessageSend().*

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

SA_AIS_ERR_INIT - The previous initialization with s*aMsgInitialize()* was incomplete, since the *saMsgMessageDeliveredCallback()* callback function is missing, and the user specified SA_MSG_MESSAGE_DELIVERED_ACK in *ackFlags* of *saMsgMessageSendAsync()*.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the provider of the service is out of memory and cannot provide the service.This error applies only to *saMsgMessageSend()*.

SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory). This error applies only to *saMsgMessageSend()*.

SA_AIS_ERR_NOT_EXIST - The destination message queue name or message queue group name, designated by *destination,* cannot be found.

SA_AIS_ERR_QUEUE_FULL - If *destination* is a message queue, it is full. If *destination* is a unicast message queue group, the selected member queue is full. If *destination* is a multicast message queue group, all selected member message queues are full.

SA_AIS_ERR_QUEUE_NOT_AVAILABLE - The *destination* parameter designates a message queue group, and the message queue group is empty.

SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

See Also

saMsgMessageSendReceive(), saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgMessageGet(), SaMsgMessageDeliveredCallbackT

3.7.2 SaMsgMessageDeliveredCallbackT

Prototype

```
typedef void (*SaMsgMessageDeliveredCallbackT)(
        SaInvocationT invocation,
        SaAisErrorT error
);
```

Parameters

invocation - [in] A designator that associates this invocation to a previous call to one of the *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* functions.

error - [in] This parameter specifies whether the message sent via the corresponding invocation of *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* has been delivered to the destination message queue, or to the reply buffer supplied by the process that invoked the *saMsgMessageSendReceive()* function.

- SA_AIS_OK - The message could be delivered 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 *msgHandle*, specified in the corresponding invocation of the *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* functions is invalid, since it is corrupted, uninitialized, or has already been finalized.
- SA_AIS_ERR_INVALID_PARAM - A parameter was not set correctly in the corresponding invocation of the *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* functions.
- SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the provider of the service is out of memory and cannot provide the service.
- SA_AIS_ERR_NO_RESOURCES - Insufficient resources (other than memory)
- SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group, designated by *destination* in the corresponding invocation of *saMsgMessageSendAsync(),* cannot be found, or the reply buffer, identified by *senderId* in the corresponding invocation of *saMsgMessageReplyAsync(),* cannot be located.
- SA_AIS_ERR_NO_SPACE - The reply buffer, identified by *senderId* in the corresponding invocation of *saMsgMessageReplyAsync(),* is not large enough to contain the reply message.
- SA_AIS_ERR_QUEUE_FULL - If the destination message queue, designated by *destination* in the corresponding invocation of the *saMsgMessageSendAsync()* function, is a message queue, it is full. If it is a unicast message queue group, the selected member message queue is full. If it is a multicast message queue group, all selected member message queues are full.

- SA_AIS_ERR_QUEUE_NOT_AVAILABLE - The *destination* parameter in the corresponding invocation of the *saMsgMessageSendAsync()* function designates a message queue group, and the message queue group is empty.
- SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter in the corresponding invocation of the *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* functions is invalid.

Description

The Message Service invokes this callback to indicate whether a previous call to *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* could deliver a message to the destination successfully. This callback is invoked in the context of a thread issuing an *saMsgDispatch()* call on the handle *msgHandle,* which was specified in the corresponding *saMsgMessageSendAsync(()* or *saMsgMessageReplyAsync()* call.

If an error occurs, it is returned in the error parameter.

During this call or after this call returns, the process may deallocate the memory for the data in the *message* buffer, which was passed previously to the corresponding *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* call.

Message delivery properties:

For *saMsgMessageSendAsync(),* the message delivery properties apply to either a destination message queue or a message queue that is a member of a destination message queue group.

- Message queue or a unicast message queue group: If *error* is SA_AIS_OK, the message has been successfully delivered to exactly one message queue; otherwise, if *error* is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been, and will not be, delivered to any destination message queue.
- Multicast message queue group: If *error* is SA_AIS_OK, the message has been successfully delivered to at least one member of the message queue group; otherwise, if *error* is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been, and will not be, delivered to any destination message queue.

For *saMsgMessageReplyAsync(),* the message delivery properties apply to the delivery of the reply message into the reply buffer supplied by the process that invoked the *saMsgMessageSendReceive()* function.
If *error* is SA_AIS_OK, the message has been successfully delivered; otherwise, if *error* is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been, and will not be, delivered.

Return Values

None.

See Also

saMsgMessageSendAsync(), saMsgMessageReplyAsync(), saMsgMessageSendReceive(), saMsgDispatch()

3.7.3 saMsgMessageGet()

Prototype

SaAisErrorT saMsgMessageGet(

SaMsgQueueHandleT queueHandle,

*SaMsgMessageT *message,*

*SaTimeT *sendTime,*

*SaMsgSenderIdT *senderId,*

SaTimeT timeout

);

Parameters

queueHandle - [in] The handle of the message queue a message is to be received from.

message - [in/out] A pointer to a message structure that contains the following fields:

- *data* - [in/out] A buffer, provided by the invoking process, for the message that is to be received. If *data* is NULL, the value of *size* provided by the invoking process is ignored, and the buffer is provided by the Message Service library. The buffer must be deallocated by the calling process by invoking the *saMsgMessageDataFree()* function.
- *size* - [in/out] The size of the buffer. After successful completion, *size* contains the size of *data.* If the *data* buffer provided by the process is too small, an error is returned, and *size* contains the size required to receive the message.
- *type* - [out] The message type.
- *version* - [out] Distinguishes different versions of a message of the same type.
- *priority* - [out] The priority of the received message.

- *senderName* [in/out] If *senderName* as an in parameter is not NULL, it points to an area to contain the sender name. If a sender name is available in the received message, the Message Service places the sender name into this area; otherwise, *message->senderName->length* is set to zero.
  If *senderName* as an in parameter is NULL, no sender name is provided to the caller.

sendTime - [in/out] If *sendTime* as an in parameter is not NULL, it points to a value of type *SaTimeT* on return from *saMsgMessageGet()*. If *saMsgMessageGet()* returns SA_AIS_OK, this value represents the absolute time when the received message was stored in the destination message queue by the calls *saMsgMessageSend()*, or *saMsgMessageSendAsync(),* or *saMsgMessageSendReceive()*. If *sendTime* as an in parameter is NULL, it is ignored.

senderId - [in/out] This parameter must point to a value of type *SaMsgSenderIdT*, defined in section 3.3.2 on page 21. If *saMsgMessageGet()* returns SA_AIS_OK and the contents of *senderId* is not zero, then the receiving thread must reply to the received message using *saMsgMessageReply()* or *saMsgMessageReplyAsync()*, and it must provide a pointer to the unmodified contents of *senderId* in this call.

timeout - [in] The time that the *saMsgMessageGet()* function waits for the arrival of a message before it issues a timeout error.
If *timeout* is set to 0, the call returns immediately; if there was a message in the message queue, and no other error occurred, this message is returned, and the return value is set to SA_AIS_OK; otherwise SA_AIS_ERR_TIMEOUT is returned.

Description

This function retrieves a message from the message queue designated by *queueHandle*. This function will block until a message is available to be retrieved, or the time specified by the *timeout* parameter elapses.

When receiving messages via an invocation of the *saMsgMessageGet()* function, messages in a higher priority area are received before messages in a lower priority area. Messages with the same priority are received in the order of their arrival in the corresponding priority area associated with a priority. For more details, see Section 3.1.6 on page 20.

When the *data* buffer is allocated by the Message Service library, care must be taken to ensure that the invoking process deallocates that buffer space by calling the *saMsgMessageDataFree()* function.

If SA_AIS_OK is returned, the received message is removed from the message queue.
If SA_AIS_ERR_LIBRARY or SA_AIS_ERR_TIMEOUT is returned, it is unspecified

whether the message is removed from the message queue or whether it is not. For all other error codes, the message is not removed from the message queue.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

- It is corrupted, was not obtained via the *saMsgQueueOpen()* or *saMsgQueueOpenCallback()* functions, or the corresponding message queue has already been closed.
- The handle *msgHandle* that was passed to the functions *saMsgQueueOpen*() or *saMsgQueueOpenAsync*() has already been finalized.

SA_AIS_ERR_INVALID_PARAM - One of the parameters is not set correctly. In particular, this value is returned if *senderId* is NULL.

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

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

SA_AIS_ERR_NO_SPACE - The message could not be received because the buffer provided was not large enough.

SA_AIS_ERR_INTERRUPT - This error code is returned if the call is terminated by a call to *saMsgMessageCancel()*.

See Also

saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgMessageSend(), saMsgMessageSendAsync(), saMsgMessageSendReceive(), saMsgMessageDataFree(), saMsgMessageCancel()

3.7.4 saMsgMessageDataFree()

Prototype

SaAisErrorT saMsgMessageDataFree(

SaMsgHandleT msgHandle,

*void *data*

);

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

data - [in] A pointer to the buffer that was allocated by the *saMsgMessageGet()* function or by the *saMsgMessageSendReceive()* function and that is to be deallocated.

Description

This function frees the memory pointed to by *data* and that was allocated by the Message Service library in a previous call to the *saMsgMessageGet()* or *saMsgMessageSendReceive()* functions.

For details, refer to the description of the *message* parameter in the corresponding invocation of the *saMsgMessageGet()* function and to the description of the *receiveMessage* parameter in the corresponding invocation of the *saMsgMessageSendReceive()* function.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saMsgMessageGet(), saMsgMessageSendReceive()

3.7.5 SaMsgMessageReceivedCallbackT

Prototype

*typedef void (*SaMsgMessageReceivedCallbackT)(*

SaMsgQueueHandleT queueHandle

);

Parameters

queueHandle - [in] The handle to the message queue from which the message can be received.

Description

The Message Service invokes this callback function to notify a process that a message can be received from the message queue, designated by *queueHandle*. This callback is invoked in the context of a thread issuing an *saMsgDispatch()* call on the handle *msgHandle,* which was specified in the *saMsgQueueOpen()* function or in the *saMsgQueueOpenAsync()* function, leading to the handle *queueHandle.*

The process can receive this message by invoking the *saMsgMessageGet()* function.

This callback is invoked whenever a message is placed in the message queue, irrespective of its priority.

Return Values

None.

See Also

saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync(), saMsgMessageSend(), saMsgMessageSendAsync(), saMsgMessageSendReceive(), saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgDispatch()

3.7.6 saMsgMessageCancel()

Prototype

SaAisErrorT saMsgMessageCancel(

SaMsgQueueHandleT queueHandle

);

Parameters

queueHandle - [in] The handle to the message queue from which a message is to be received.

Description

This function cancels all blocking calls to *saMsgMessageGet()* for the message queue, designated by *queueHandle,* in the invoking process.

This function is normally called during fault recovery.

The canceled call returns with the error code set to SA_AIS_ERR_INTERRUPT. If no process is blocking in an *saMsgMessageGet()* call, the *saMsgMessageCancel()* call has no effect and returns SA_AIS_ERR_NOT_EXIST.

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

- It is corrupted, was not obtained via the *saMsgQueueOpen()* or *saMsgQueueOpenCallback()* functions, or the corresponding message queue has already been closed.
- The handle *msgHandle* that was passed to the functions *saMsgQueueOpen*() or *saMsgQueueOpenAsync*() has already been finalized.

SA_AIS_ERR_NOT_EXIST - No process was blocking in an *saMsgMessageGet()* call.

See Also

saMsgMessageGet()

3.8 Request-Reply Operations

3.8.1 saMsgMessageSendReceive()

Prototype

```
SaAisErrorT saMsgMessageSendReceive(
    SaMsgHandleT msgHandle,
    const SaNameT *destination,
    const SaMsgMessageT *sendMessage,
    SaMsgMessageT *receiveMessage,
    SaTimeT *replySendTime,
    SaTimeT timeout
);
```

Parameters

msgHandle - [in] The handle, obtained through the *saMsgInitialize()* function, designating this particular initialization of the Message Service.

destination - [in] A pointer to the name of a message queue or a unicast message queue group the message, designated by *sendMessage,* is to be sent to. It is not permitted to specify a multicast message queue group in *destination*.

sendMessage - [in] A pointer to the message structure for the message to be sent, consisting of a *type* field that contains the message type, a *version* field that is used to distinguish different versions of a message of the same type, a *data* field for the message to be sent, a *size* field that contains the size of the message, a *priority* field that gives the priority of the message to be sent, and a *senderName* pointer. If *senderName* is not NULL, it points to an area containing the sender name, which is supplied by the caller; If *senderName* is NULL, the sending process does not provide its sender name, and a receiving process expecting a sender name will get *message->senderName->length* set to zero.

receiveMessage - [in/out] A pointer to the message structure that contains the following fields:

- *data* - [in/out] A buffer, provided by the invoking process, for the reply message to be received. This buffer is called reply buffer. If *data* is NULL, the value of *size* provided by the invoking process is ignored, and the reply buffer is provided by the Message Service library. The reply buffer must be deallocated by the calling process by invoking the *saMsgMessageDataFree()* function.
- *size* - [in/out] The size of the reply buffer. After successful completion, the *size* field contains the size of the data in the message. If the *data* buffer provided by the process is too small, an error is returned.
- *type* -[out] The message type.
- *version* - [out] Distinguishes different versions of a message of the same type.
- *priority* - [out] The priority of the received message.
- *senderName* [in/out] If *senderName* as an in parameter is not NULL, it points to an area to contain the sender name of the process replying to this *saMsgMessageSendReceive()* call. If a sender name is available in the received message, the Message Service places the sender name into this area; otherwise, *receiveMessage->senderName->length* is set to zero.
  If *senderName* as an in parameter is NULL, no sender name is provided to the caller.

replySendTime - [in/out] If *replySendTime* as an in parameter is not NULL, it points to a value of type *SaTimeT* on return from *saMsgMessageSendReceive().* This value represents the timestamp when the reply message was written into the *data* field of the structure pointed to by the *receiveMessage* parameter. If *replySendTime* as an in parameter is NULL, it is ignored.

timeout -[in] The time that the *saMsgMessageSendReceive()* function must wait before issuing a timeout error.

Description

This function enables the transmission of a message, designated by *sendMessage,* as well as the receipt of a reply message, designated by *receiveMessage,* in a single invocation. It sends the message, designated by *sendMessage,* to the message queue or message queue group, designated by *destination*. The message is sent synchronously, i.e., the process blocks waiting for a reply. The reply consists of the *receiveMessage* message. It must be sent using either the *saMsgMessageReply()* function or the *saMsgMessageReplyAsync()* function, and must arrive within a time

interval specified by the *timeout* parameter; otherwise, the error SA_AIS_ERR_TIMEOUT is returned.

In absence of errors, the thread that invoked *saMsgMessageSendReceive()* will receive the corresponding reply, even when other threads are waiting for replies to other *saMsgMessageSendReceive()* invocations. Moreover, the reply message, sent via the *saMsgMessageReply()* or *saMsgMessageReplyAsync()* functions, is not enqueued in a message queue.

After this call returns, the invoking process may deallocate the memory for the data in *sendMessage.* When the *data* buffer referred to by the *receiveMessage* structure is allocated by the Message Service library, care must be taken to ensure that the invoking process deallocates that buffer space promptly by calling the *saMsgMessageDataFree()* function.

Message delivery properties:

These properties apply when a message is sent to either a destination message queue or to a message queue that is a member of a destination message queue group.

If SA_AIS_ERR_QUEUE_FULL is returned, then the message has not been delivered to the process receiving from the destination message queue, because the message queue was full.

If SA_AIS_ERR_NO_SPACE is returned, then the message has been delivered to a process receiving from the destination message queue, because this error applies only when receiving the reply.

If SA_AIS_ERR_TIMEOUT, SA_AIS_ERR_NO_MEMORY or SA_AIS_ERR_NO_RESOURCES is returned, the message may or may not have been delivered to a process receiving from the destination message queue, because those errors apply to both the sending and receiving parts of this call.

For all other return values except SA_AIS_OK, the Message Service guarantees that no process receiving from the destination message queue receives this message.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - One of the parameters is not set correctly. In particular, this applies if a multicast message queue group is specified in the *destination* parameter.

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

SA_AIS_ERR_NO_RESOURCES - There are not enough resources (other than memory).

SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group, designated by *destination,* cannot be found.

SA_AIS_ERR_NO_SPACE - The message could not be received because the reply buffer, provided by the invoking process, is not large enough.

SA_AIS_ERR_QUEUE_FULL - If *destination* is a message queue, it is full. If *destination* is a unicast message queue group, the selected member message queue is full.

SA_AIS_ERR_QUEUE_NOT_AVAILABLE - The *destination* parameter designates a message queue group, and the message queue group is empty.

See Also

saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgMessageGet(), saMsgMessageSend(), saMsgMessageSendAsync(), saMsgMessageDataFree()

3.8.2 saMsgMessageReply() and saMsgMessageReplyAsync()

Prototype

```
SaAisErrorT saMsgMessageReply(
        SaMsgHandleT msgHandle,
        const SaMsgMessageT *replyMessage,
        const SaMsgSenderIdT *senderId,
        SaTimeT timeout
);
SaAisErrorT saMsgMessageReplyAsync(
        SaMsgHandleT msgHandle,
        SaInvocationT invocation,
        const SaMsgMessageT *replyMessage,
        const SaMsgSenderIdT *senderId,
        SaMsgAckFlagsT ackFlags
);
```

Parameters

msgHandle - [in] The handle, obtained through the saMsgInitialize() function, designating a particular initialization of the Message Service.

invocation - [in] This parameter associates this invocation of *saMsgMessageReplyAsync()* with a corresponding invocation of the *saMsgMessageDeliveredCallback()* function. This parameter is ignored if *ackFlags* is set to zero, meaning that the *saMsgMessageDeliveredCallback()* function is not called, and the caller is not informed whether an error occurred or whether it did not.

replyMessage - [in] A pointer to a structure, defined by the *SaMsgMessageT* in Section 3.3.11 on page 28, for the reply message to be sent that consists of the *type, version, data, size*, *priority,* and *senderName* fields. The *priority* field is not used and is set to zero by the Message Service. If *senderName* is not NULL, it points to an area containing the sender name, which is supplied by the caller; if *senderName* is NULL, the replying process does not provide its sender name, and the receiving process (that is the process waiting for a reply in the *saMsgMessageSendReceive()* call) expecting a sender name will get a *receiveMessage->senderName->length* set to zero.

senderId - [in] The value that the *saMsgMessageGet()* call received in the *senderId* field for the message that the caller is replying to.

ackFlags - [in] The kind of the required acknowledgment. This field must be set to zero or to SA_MSG_MESSAGE_DELIVERED_ACK. In the latter case, the caller requires to be acknowledged whether the message can be stored in the reply buffer provided in the corresponding *saMsgMessageSendReceive()* call. If there is no space for the entire message in the reply buffer, the error SA_AIS_ERR_NO_SPACE is returned.

timeout - [in] The *saMsgMessageReply()* invocation is considered to have failed if it does not complete within the duration specified.

Description

These functions are used to reply to a message that was sent using the *saMsgMessageSendReceive()* function in which case the Message Service sets the *senderId* for the message received by the *saMsgMessageGet()* function to a non-zero value. In this case, the *saMsgMessageReply()* or *saMsgMessageReplyAsync()* function must be used to reply to the received message, and they must supply as the *senderId* parameter the same value as the *senderId* field, obtained through the *saMsgMessageGet()* function.

A process may not reply to a message more than once.

The *saMsgMessageReply()* function waits synchronously (that is, it blocks) until the reply message has been placed in the reply buffer provided by the process that invoked *saMsgMessageSendReceive()*.
After the *saMsgMessageReply()* function returns, the invoking process may deallocate the memory for the data in *replyMessage.*

The function *saMsgMessageReplyAsync()* returns as soon as possible, without waiting until the reply message has been placed in the reply buffer provided by the process that invoked *saMsgMessageSendReceive().* If the value of the *ackFlags* field is zero, the *saMsgMessageDeliveredCallback()* is not invoked, and the caller is not informed if an error occurs. If the value of the *ackFlags* field is set to SA_MSG_MESSAGE_DELIVERED_ACK, and this call returns SA_AIS_OK, *saMsgMessageDeliveredCallback()* is invoked to indicate whether the message was delivered, or whether an error occurred. For this purpose, the user must have supplied the *saMsgMessageDeliveredCallback()* when invoking the *saMsgInitialize()* function.
If *saMsgMessageReplyAsync()* returns successfully, and *ackFlags* is not set to zero, the sending process may deallocate the memory for the data in the message buffer either during the invocation of *saMsgMessageDeliveredCallback()* or after *saMsgMessageDeliveredCallback()* returns.

If *saMsgMessageReplyAsync()* returns an error, or if *ackFlags* is set to zero, meaning that *saMsgMessageDeliveredCallback()* will not be called, the process may deallocate the memory for the data in the message buffer as soon as *saMsgMessageReplyAsync()* returns.

If the process that invoked *saMsgMessageSendReceive()* exits or calls *saMsgFinalize()* for its handle *msgHandle* before *saMsgMessageReply()* or *saMsgMessageReplyAsync()* completes, the Message Service returns SA_AIS_ERR_NOT_EXIST to the process that invoked *saMsgMessageReply()* or *saMsgMessageReplyAsync().*

Message delivery properties:

saMsgMessageReply:
If the return value is SA_AIS_OK, the Message Service delivers the reply to the process that invoked *saMsgMessageSendReceive()*. If the return value is SA_AIS_ERR_LIBRARY, or SA_AIS_ERR_TIMEOUT, it is unspecified whether the reply is delivered or not. In all other cases, the Message Service shall not deliver the reply.

saMsgMessageReplyAsync():
If *saMsgMessageReplyAsync()* returns SA_AIS_OK, and if the value of the *ackFlags* field is set to SA_MSG_MESSAGE_DELIVERED_ACK, the Message Service will invoke *saMsgMessageDeliveredCallback()*; otherwise, if the error code is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the Message Service will not invoke *saMsgMessageDeliveredCallback(),* and it will not deliver the reply. Refer to the *saMsgMessageDeliveredCallback()* function for the message delivery properties.

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 - Before the call could complete, either an implementation-dependent timeout occurred, or the timeout specified by the *timeout* parameter in the *saMsgMessageReply()* call occurred. It is unspecified whether the *saMsgMessageReply()* 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 *msgHandle* is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_INIT - The previous initialization with s*aMsgInitialize()* was incomplete, since the *saMsgMessageDeliveredCallback()* callback function is missing, and the

user specified SA_MSG_MESSAGE_DELIVERED_ACK in *ackFlags* of *saMsgMessageReplyAsync()*.

SA_AIS_ERR_INVALID_PARAM - One of the parameters is not set correctly, or the message being replied to was not sent using *saMsgMessageSendReceive()*.

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

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

SA_AIS_ERR_NOT_EXIST - Either there is no thread waiting for a reply, or the *senderId* parameter is invalid.

SA_AIS_ERR_NO_SPACE - The reply buffer, provided in the corresponding *saMsgMessageSendReceive()* call, is not large enough for the reply message.

SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

See Also

saMsgMessageSendReceive(), saMsgMessageGet(), saMsgMessageSend(), saMsgMessageSendAsync()

4 Alarms and Notifications

The Message Service produces certain alarms and notifications in order to convey important information regarding its operational and functional state to an administrator or a management system.

These reports vary in perceived severity and include alarms, which potentially require an operator intervention and notifications that signify important state or object changes. A management entity should regard notifications, but they do not necessarily require an operator intervention.

The recommended vehicle to be used for producing alarms and notifications is the Notification Service of the Service Availability™ Forum (abbreviated to NTF, see [2]), and hence the various notifications are partitioned into categories as described in this service.

In some cases, this specification uses the word “Unspecified” for values of attributes, which the vendor is at a liberty to set to whatever makes sense in the vendor’s context, and the SA Forum has no specific recommendation regarding such values. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [4]).

4.1 Setting Common Attributes

The tables presented in Section 4.2 refer to the attributes in the following list, but do not describe them, as these attributes are described in the list in a generic manner. For each attribute in this list, the specification provides recommendations regarding how to populate the attribute.

- Correlation Ids - They are supplied to correlate two notifications that have been generated because of a related cause. This attribute is optional. But in case of alarms that are generated to clear certain conditions, i.e., produced with a perceived severity of *SA_NTF_SEVERITY_CLEARED*, the correlation id shall be populated by the application with the notification Id that was generated by the Notification Service while invoking the *saNtfNotificationSend()* API during the production of the actual alarm.
- Event Time - The application might pass a timestamp or optionally pass an *SA_TIME_UNKNOWN* value in which case the timestamp is provided by the Notification Service.
- NCI Id - The *vendorId* portion of the *SaNtfClassIdT* data structure must be set to *SA_NTF_VENDOR_ID_SAF* always. The *majorId* and *minorId* will vary based on the specific SA Forum service and the particular notification. Every SA Forum

service shall have a *majorId* as described in the enumeration *SaNtfSafServicesT* of the Notification Service specification.

- Notification Id - This attribute is obtained from the Notification Service when a notification is generated, and hence need not be populated by an application.
- Notifying Object - DN of the entity generating the notification. This name must conform to the SA Forum AIS naming convention and contain at least the *safApp* RDN value portion of the DN set to the specified standard RDN value of the SA Forum AIS service generating the notification. For details on the SA Forum AIS naming convention, refer to the SA Forum Overview document.

4.2 Message Service Notifications

The following sections describe a set of notifications that a Message Service implementation shall produce.

The value of the *majorId* field in the notification class identifier (*SaNtfClassIdT)* should be set as follows in all notifications generated by the Message Service.

- *majorId* = SA_SVC_MSG

The *minorId* field within the notification class identifier (*SaNtfClassIdT)* is set distinctly for each individual notification as described below. This field is range-bound, and the used ranges are:

- Alarms: (0x01 - 0x64)
- State change notifications: (0x65 - 0xC8)
- Object change notifications: (0xC9 - 0x12C)
- Attribute change notifications: (0x12D - 0x190)

4.2.1 Message Service Alarms

4.2.1.1 Message Service impaired

Description

The Message Service is currently unable to provide service or is in a degraded state because of certain issues with memory, resources, communication, or other constraints.

Clearing Method

1) Manual, after taking appropriate administrative action or

2) Issue an implementation specific optional alarm with perceived severity *SA_NTF_SEVERITY_CLEARED* to convey that the Message Service self-healed/ recovered and is again providing service.

NTF Attribute Name	Attribute Type (X.73Y Recommendation or NTF)	SA Forum Recommended Value
Event Type	Mandatory	*SA_NTF_ALARM_COMMUNICATION*
Notification Object	Mandatory	MSG service, same as Notifying object as specified above.
Notification Class Identifier	NTF internal	*minorId* = 0x01
Additional Text	Optional	“MSG service impaired.”
Additional Information ID	Optional	Unspecified
Probable Cause	Mandatory	Applicable value from enum *SaNtfProbableCauseT* in [2]
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Applicable value from enum *SaNtfSeverityT* in [2]
Trend Indication	Optional	Unspecified
Threshold Information	Optional	Unspecified
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

4.2.2 Message Service State Change Notifications

4.2.2.1 Message Queue Capacity Reached

Description

All priority areas of the message queue are filled up to their critical capacities (see Section 3.3.12 on page 29).

NTF Attribute Name	Attribute Type (X.73Y Recommendation or NTF)	SA Forum Recommended Value
Event Type	Mandatory	*SA_NTF_OBJECT_STATE_CHANGE*
Notification Object	Mandatory	LDAP DN of the message queue, which is full and cannot accept any more messages.
Notification Class Identifier	NTF internal	*minorId* = 0x65
Additional Text	Optional	Unspecified
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION* or *SA_NTF_UNKNOWN_OPERATION*
Changed State Attribute ID	Optional	*SA_MSG_DEST_CAPACITY_STATUS*
Old Attribute Value	Optional	Unspecified
New Attribute Value	Mandatory	*SA_MSG_QUEUE_CAPACITY_REACHED*

4.2.2.2 Message Queue Capacity Available

Description

At least one priority area of the message queue is no longer filled up to its critical capacity after the Message Queue Group Capacity Reached condition has been notified for the message queue group before (see Section 3.3.12 on page 29).

NTF Attribute Name	Attribute Type (X.73Y Recommendation or NTF)	SA Forum Recommended Value
Event Type	Mandatory	*SA_NTF_OBJECT_STATE_CHANGE*
Notification Object	Mandatory	LDAP DN of the message queue, which is available for receipt of messages.
Notification Class Identifier	NTF internal	*minorId* = 0x66
Additional Text	Optional	Unspecified
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION* or *SA_NTF_UNKNOWN_OPERATION*
Changed State Attribute ID	Optional	*SA_MSG_DEST_CAPACITY_STATUS*
Old Attribute Value	Optional	*SA_MSG_QUEUE_CAPACITY_REACHED*
New Attribute Value	Mandatory	*SA_MSG_QUEUE_CAPACITY_AVAILABLE*

4.2.2.3 Message Queue Group Capacity Reached

Description

All priority areas of all the message queues within a message queue group are filled up to their critical capacities (see Section 3.3.12 on page 29).

NTF Attribute Name	Attribute Type (X.73Y Recommendation or NTF)	SA Forum Recommended Value
Event Type	Mandatory	*SA_NTF_OBJECT_STATE_CHANGE*
Notification Object	Mandatory	LDAP DN of the message queue group, which is full and cannot accept any more messages.
Notification Class Identifier	NTF internal	*minorId* = 0x67
Additional Text	Optional	Unspecified
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION* or *SA_NTF_UNKNOWN_OPERATION*
Changed State Attribute ID	Optional	*SA_MSG_DEST_CAPACITY_STATUS*
Old Attribute Value	Optional	Unspecified
New Attribute Value	Mandatory	*SA_MSG_QUEUE_GROUP_CAPACITY_REACHED*

4.2.2.4 Message Queue Group Capacity Available

Description

At least one priority area in one message queue in the message queue group is no longer filled up to its critical capacity after the Message Queue Group Capacity Reached condition has been notified for the message queue group before (see Section 3.3.12 on page 29).

NTF Attribute Name	Attribute Type (X.73Y Recommendation or NTF)	SA Forum Recommended Value
Event Type	Mandatory	*SA_NTF_OBJECT_STATE_CHANGE*
Notification Object	Mandatory	LDAP DN of the message queue group, which is available to receive messages.
Notification Class Identifier	NTF internal	*minorId* = 0x68
Additional Text	Optional	Unspecified
Additional Information ID	Optional	Unspecified
Source Indicator	Mandatory	*SA_NTF_OBJECT_OPERATION* or *SA_NTF_UNKNOWN_OPERATION*
Changed State Attribute ID	Optional	*SA_MSG_DEST_CAPACITY_STATUS*
Old Attribute Value	Optional	*SA_MSG_QUEUE_GROUP_CAPACITY_REACHED*
New Attribute Value	Mandatory	*SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE*