Service Availability^{TM} Forum Application Interface Specification

Message Service SAI-AIS-MSG-B.03.01

.

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

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

1.2 AIS Documents Organization

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

1.3 History

Previous releases of the Message Service specification:

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

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

(3) SAI-AIS-MSG-B.02.01

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

1.3.1 New Topics

- Section 3.1.1 introduces the notion of message metadata. This implies clarifications in Section 3.4.5.2 and Section 3.4.8.1.
- Section 3.2 describes the behavior of the Message Service API on a cluster node that is not in the cluster membership (see [4]).
- Section 3.4.12.2 introduces the *SaMsgQueueThresholdsT* type, which is used to set and retrieve critical capacity thresholds for the priority areas of a message queue.

- Section 3.4.13 describes the *SaMsgLimitIdT* enum, which provides a set of values that identify limits for a particular implementation of the Message Service.
- Section 3.10 describes the set and get capacity thresholds operations.
- Section 3.11 describes functions to retrieve implementation-specific values (message metadata size and a particular implementation limit).
- Chapter 4 presents the Message Service UML Information Model. The Message Service UML classes (see FIGURE 1 on page 100) were previously contained in [1]. Two new attributes have been added and the constraints of some attributes have changed.
- Chapter 5 states that no administration APIs are provided for the Message Service.
- Chapter 7 presents the Message Service Management Interface.

1.3.2 Clarifications

- See also Section 1.3.1 and Section 1.3.3.
- Section 3.1.3.2 clarifies that an application must not expect that the Message Service preserves messages in case of node failures.
- Section 3.5.3 on the *saMsgDispatch()* function clarifies the meaning of the SA_AIS_OK return value.
- The description of the *saMsgFinalize()* function (see Section 3.5.4) clarifies that this function frees all resources allocated by the Message Service for the process in this association between the process and the Message Service.
- The description of the *saMsgQueueClose()* function (see Section 3.6.3) clarifies which resources this function frees for the invoking process.
- The return values subsection of Section 3.7.5 on the *saMsgQueueGroupTrack()* function clarifies when the SA_AIS_ERR_INIT is returned.
- Section 6.2 clarifies the setting of the notifying object in notifications.

1.3.3 Changes in Return Values of API Functions:

Table 1 Changes in Return Values of API Functions

API Function	Return Value	Change Type
All API functions except *saMsgFinalize()* and *SaMsgMessageReceivedCallbackT*	SA_AIS_ERR_UNAVAILABLE	new
saMsgQueueOpen(), *saMsgQueueOpenAsync()*, *SaMsgQueueOpenCallbackT*, *saMsgMessageSend()*, *saMsgMessageSendAsync()*, *SaMsgMessageDeliveredCallbackT*, *saMsgMessageSendReceive()*, *saMsgMessageReplyAsync()*, and *saMsgMessageReplyAsync()*	SA_AIS_ERR_TOO_BIG	new
saMsgQueueOpen(), *saMsgQueueOpenAsync()*, and *SaMsgQueueOpenCallbackT*	SA_AIS_ERR_NO_RESOURCES	extended
SaMsgQueueGroupTrackCallbackT	SA_AIS_ERR_INVALID_PARAM	clarified
saMsgMessageSend() and *saMsgMessageSendAsync()*	SA_AIS_ERR_NO_MEMORY SA_AIS_ERR_NO_RESOURCES	extended

1.3.4 Removed Topics

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

SA Forum does not mandate that Message Service implementations which also support the B.02.01 version must generate the "service impaired” alarm for the B.02.01 version.

The "service impaired” alarm has also been removed from the Message Service MIB for the Message Service B.03.01 version.

1.3.5 Other Changes

- Section 3.1.1 and Section 3.1.2 state now that message queues and message queue groups respectively cannot be configured statically. Only a dynamic configuration is provided.
- The attribute *saMsgQueueGroupMemberName* has been added to the runtime object class *SaMsgQueueGroup* (refer to FIGURE 1 in Section 4.2). This attribute was missing in the SAI-AIS-MSG-B.02.01 version of the Message Service.

1.4 References

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

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

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

[3] Service Availability^{TM} Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.01.01

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

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

[6] Service Availability^{TM} Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.02.01

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

References to these documents are made by placing 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 message queues and read from them. 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 by 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 1 receiver whereas with message queue groups, the Message Service uses the model of *m* senders to *n* receivers.

1. The term “node” unless otherwise qualified is used in this document in the sense of a “member node”, as defined in the Cluster Membership Service specification (see [4]).

3 SA Message Service API

3.1 Message Service Model

3.1.1 Messages and Message Queues

A **message** consists of message data plus associated metadata.
The **message data** consists of a byte array of a certain size. The contents of the message data are opaque to the Message Service.
The **message metadata** comprises a **standard portion** and an **implementation-specific** portion. The standard metadata portion of a message covers its **type**, **version**, **priority**, **data size**, variable-length **sender name** (rounded up to the next multiple of 8 bytes), and sender name length. These terms are explained in Section 3.4.11 on page 30.
The implementation-specific metadata portion has a fixed size for any message.
The term **message size** denotes the sum of the size of the message data (rounded up to the next multiple of 8 bytes) and the size of the message metadata.

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.

Message queues can be dynamically created and deleted.

3.1.2 Message Queue Groups

A **message queue group** is a collection of message queues that are addressed as a single entity. A message queue can be a member of more than one message queue group. If a message queue is removed from the cluster, the message queue is automatically removed from all message queue groups of which it is a member.

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.

Message queue groups can be dynamically created and deleted.

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. Hence, more than one process can receive the message.

An implementation can support several different unicast and multicast **policies** described below. 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 selects the next member in turn and for how many members this procedure 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 no local member exists, 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 whether the Message Service selects the next member in turn and for how many members this procedure 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 no local member exists, 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 whether the Message Service selects the next member in turn and for how many members this procedure is repeated.

- **Broadcast** (multicast)

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

3.1.3 Properties of Message Queues

3.1.3.1 Nonpersistent and Persistent Message Queues

A message queue can be defined as **nonpersistent**, meaning that the Message Service removes the message queue automatically from the cluster-wide name space if no process has opened this message queue 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 nonpersistent message queue with an infinite retention time. A persistent message queue can be removed only by an explicit call to the *saMsgQueueUnlink()* function.

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.

3.1.3.2 Message Preservation Property of a Queue

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

In node switch-over situations, the Message Service must preserve messages in a message queue which have not yet been consumed. For example, assume a service unit (for details, refer to [6]) containing a component *cx* that 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 which is located on another node and which contains the component *cy* that works as standby for *cx*'s service. If *cy* reopens the queue *Q* and 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, an application must not expect that the Message Service preserves messages in case of node failures.

3.1.4 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.

3.1.5 Message Delivery Properties

- **Priority** - When a process receives a message from a message queue by invoking *saMsgMessageGet()*, the process receives messages in a higher priority area before it receives messages in a lower priority area. The process receives messages of the same priority from a sending process in the order in which that process sent them. It might not receive messages of different priorities from a sending 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. A message sent to a unicast message queue group is delivered at most once to one of the member message queues. A message sent to a multicast message queue group is delivered at most once to each of the selected member message queues.
- **Delivery guarantees** - If a process sends a message to a message queue or to a unicast message queue group, and the space in the destination message queue is not enough to hold the entire message, the error code SA_AIS_ERR_QUEUE_FULL is returned, assuming that the process requested an acknowledgment for its send operation. It is expected that for correctly constructed sending calls, an implementation 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 but only a SA_AIS_ERR_TIMEOUT timeout error.
  If a process sends a message to a multicast message queue group and

requests an acknowledgment, SA_AIS_OK is returned 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 for an acknowledgment that the sent message has been stored in the destination message queue (refer to send operations in Section 3.8.1) or that the reply to a sending process has been received by the sending process (refer to request/reply operations in Section 3.9).
- **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 by invoking *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, and so on. 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.
  See also Section 3.1.3.2.

3.2 Unavailability of the Message Service API on a Non-Member Node

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

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

3.2.1 A Member Node Leaves or Rejoins the Cluster Membership

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

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

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

- The functions *saMsgQueueOpenAsync()* and *saMsgQueueGroupTrack()* (assuming that the latter function requires a callback) may return SA_AIS_OK or SA_AIS_ERR_UNAVAILABLE, depending on the service implementation. If they return SA_AIS_OK, the callbacks *SaMsgQueueOpenCallbackT* and *SaMsgQueueGroupTrackCallbackT* respectively will be called and will also return SA_AIS_ERR_UNAVAILABLE in the *error* parameter; otherwise, the callbacks will not be called.
- If the *saMsgMessageSendAsync()* and *saMsgMessageReplyAsync()* API functions require an acknowledgment, they may return SA_AIS_OK or SA_AIS_ERR_UNAVAILABLE, depending on the service implementation. If they return SA_AIS_OK, *SaMsgMessageDeliveredCallbackT* will be called and will also return SA_AIS_ERR_UNAVAILABLE in the *error* parameter; otherwise, *SaMsgMessageDeliveredCallbackT* will not be called.
- The *saMsgFinalize()* function, which is used to free the library handles and all resources associated with these handles.

⇒ Any outstanding callbacks *SaMsgQueueOpenCallbackT*, *SaMsgQueueGroupTrackCallbackT*, and *SaMsgMessageDeliveredCallbackT* will return SA_AIS_ERR_UNAVAILABLE in the *error* parameter.

⇒ The callback *SaMsgMessageReceivedCallbackT* will not be called.

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

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

3.2.2 Guidelines for Message Service Implementers

The implementation of the Message Service must leverage the SA Forum Cluster Membership Service (see [4]) to determine the membership status of a node for the case explained in Section 3.2.1 before returning SA_AIS_ERR_UNAVAILABLE. If the Cluster Membership Service considers a node as a member of the cluster but the

Message Service experiences difficulty in providing service to its clients because of transport, communication, or other issues, it must respond with SA_AIS_ERR_TRY_AGAIN.

3.3 Include File and Library Name

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

#include <saMsg.h>

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

libSaMsg.so

3.4 Type Definitions

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

3.4.1 Handles

3.4.1.1 SaMsgHandleT

typedef SaUint64T SaMsgHandleT;

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

3.4.1.2 SaMsgQueueHandleT

typedef SaUint64T SaMsgQueueHandleT;

This type is used for the handle to a message queue.

3.4.2 SaMsgSenderIdT

typedef SaUint64T SaMsgSenderIdT;

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

3.4.3 SaMsgCallbacksT

The *SaMsgCallbacksT* structure is defined as follows:

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

A structure of the *SaMsgCallbacksT* type (called a callbacks structure) is used to specify the callback functions that the Message Service may invoke.

3.4.4 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 be set to either 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 that the message has been stored in the destination message queue or **reply buffer** (see Section 3.8.2 on page 74). If the space in the destination message queue or reply buffer is not enough for the entire message, the Message Service returns SA_AIS_ERR_QUEUE_FULL in case of a message queue and SA_AIS_ERR_NO_SPACE in case of a reply buffer.
If SA_MSG_MESSAGE_DELIVERED_ACK is not set, the caller does not require an acknowledgment.

3.4.5 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.

3.4.5.1 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 the SA_MSG_QUEUE_PERSISTENT flag is not set, the message queue is nonpersistent, meaning that the Message Service removes the message queue automatically if it is not opened by a process after the message queue is closed and before the retention time of the message queue elapses.

3.4.5.2 SaMsgQueueCreationAttributesT

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

The fields of the *SaMsgQueueCreationAttributesT* type 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. Both message data and message metadata consume space in the priority area. For the meaning of SA_MSG_MESSAGE_LOWEST_PRIORITY, refer to Section 3.4.7 on page 26. The size of the implementation-specific message metadata can be obtained by invoking the *saMsgMetadataSizeGet()* function (see Section 3.11.1 on page 95).
- *retentionTime* - The time duration after a process closes the message queue until the Message Service removes the message queue. This parameter applies only to nonpersistent message queues.

3.4.6 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 the Message Service to create a message queue if the message queue does not exist.
- SA_MSG_QUEUE_RECEIVE_CALLBACK - This flag requests the Message Service to notify the process about the arrival of messages by invoking the *saMsgMessageReceivedCallback()* callback 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 process can still invoke 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 the flag is not set, the Message Service must preserve existing messages when opening the message queue. However, the Message Service cannot guarantee that messages will be preserved in a fail-over situation.

3.4.7 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
```

3.4.8 Message Queue Usage and Status

This section defines the types *SaMsgQueueUsageT* and *SaMsgQueueStatusT*, which are used to obtain information about a message queue.

3.4.8.1 SaMsgQueueUsageT

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

The fields of the *SaMsgQueueUsageT* type 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. This size is specified when the corresponding queue is created. This space is used to hold messages of a particular priority and is consumed by both message data and message metadata.
- *queueUsed* - The current number of bytes in the priority area occupied by messages of a particular priority. This number includes both message data and message metadata.
- *numberOfMessages* - The current number of messages in the priority area.

3.4.8.2 SaMsgQueueStatusT

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

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

- *creationFlags* - Zero or SA_MSG_QUEUE_PERSISTENT, which was defined in Section 3.4.5 on page 24.
- *retentionTime* - The time duration after a process closes the message queue until the Message Service removes it. This field applies only to nonpersistent message queues.

- *closeTime* - The absolute time when the message queue was last closed. This field contains zero if the message queue is currently open for a process.
- *saMsgQueueUsage* - An array containing the message queue usage data for each priority area of the message queue.

3.4.9 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 3.1.2 on page 17.

3.4.10 Types for Tracking Message Queue Group Changes

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

3.4.10.1 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 (as defined in Section 3.7.5) 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.

3.4.10.2 SaMsgQueueGroupMemberT

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

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

3.4.10.3 SaMsgQueueGroupNotificationT

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

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

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

3.4.10.4 SaMsgQueueGroupNotificationBufferT

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

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

- *numberOfItems* - Number of elements in the array to which *notification* points. Each element in this array is of type *SaMsgQueueGroupNotificationT*.
- *notification* - Pointer to the notification array.
- *queueGroupPolicy* - The load distribution policy of the message queue group.

3.4.11 SaMsgMessageT

This type describes a message to be used for sending and receiving.

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

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

- *type* - Message type that is specified by a process when it sends a message.
- *version* - Version of the message. This field is used to distinguish different versions of messages with the same message type. It is the responsibility of the application program to set *version* and to ensure that the application program can handle messages with appropriate different versions.
- *size* - Size of the message data in bytes, which is set by a process when it sends a message and by the Message Service when it receives a message.

- *senderName* - A pointer to a name that 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 3.4.7 on page 26.

3.4.12 Critical Capacity of Message Queues and Message Queue Groups

The Message Service defines the 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 6 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 consists of a pair of values, one for the high-water capacity threshold and the other for the low-water capacity threshold. The status of a priority area changes to critical if the usage of the priority area exceeds the **high-water capacity threshold** value. The status of a priority area changes to non-critical if the status of a priority area is already critical and the usage falls below the **low-water capacity threshold** value.
The high-water and low-water capacity thresholds must be lesser than or equal to the priority area size.

The user can set the critical capacity of the priority areas of a queue by invoking the *saMsgQueueCapacityThresholdsSet()* function. If not set by the *saMsgQueueCapacityThresholdsSet()* function, the low-water and the high-water capacity thresholds of a priority area are identical to the corresponding priority area size.

The user can retrieve the critical capacity of a queue by invoking the *saMsgQueueCapacityThresholdsGet()* function.

3.4.12.1 saMsgMessageCapacityStatusT

The following enumeration contain values representing 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.

```
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 again 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 fact is potentially affecting the capability of the message queue group to accept new messages.
- SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE - At least one priority area in one message queue within the queue group is no longer filled up to its critical capacity and is available to again accept new messages after having recovered from a previous SA_MSG_QUEUE_GROUP_CAPACITY_REACHED condition.

3.4.12.2 SaMsgQueueThresholdsT

```
typedef struct {
        SaSizeT capacityReached[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];
        SaSizeT capacityAvailable[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];
} SaMsgQueueThresholdsT;
```

This type is used to set and to retrieve critical capacity thresholds for the priority areas of a message queue. The values set for each priority area *i* must fulfill the following relationships:

0 <= *capacityAvailable*[*i*] <= *capacityReached*[*i*] <= *size*[*i*],

where *size* represents the array containing the sizes of the priority areas as specified at creation time, and *capacityReached* and *capacityAvailable* are arrays that contain for each priority area the high-water capacity threshold and the low-water capacity threshold respectively.

3.4.12.3 saMsgStateT

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

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

3.4.13 SaMsgLimitIdT

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

The user can retrieve at runtime the current value of a particular limit by specifying the identifier of the limit (one of the enum values of the type *SaMsgLimitIdT*, defined below) when invoking the *saMsgLimitGet()* function (see Section 3.11.2 on page 97).

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

```
typedef enum {
        SA_MSG_MAX_PRIORITY_AREA_SIZE_ID            = 1,
        SA_MSG_MAX_QUEUE_SIZE_ID                    = 2,
        SA_MSG_MAX_NUM_QUEUES_ID                    = 3,
        SA_MSG_MAX_NUM_QUEUE_GROUPS_ID              = 4,
        SA_MSG_MAX_NUM_QUEUES_PER_GROUP_ID          = 5,
        SA_MSG_MAX_MESSAGE_SIZE_ID                  = 6,
        SA_MSG_MAX_REPLY_SIZE_ID                    = 7
} SaMsgLimitIdT;
```

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

- SA_MSG_MAX_PRIORITY_AREA_SIZE_ID - This enum can be used to retrieve the maximum size in bytes of a single priority area of a message queue.
- SA_MSG_MAX_QUEUE_SIZE_ID - This enum can be used to retrieve the maximum cumulative size in bytes of all priority areas of a message queue. This value can be smaller, equal, or larger than the number of priority areas multiplied by SA_MSG_MAX_PRIORITY_AREA_SIZE_ID.
- SA_MSG_MAX_NUM_QUEUES_ID - This enum can be used to retrieve the cluster-wide maximum number of queues that can exist at the same time.
- SA_MSG_MAX_NUM_QUEUE_GROUPS_ID - This enum can be used to retrieve the cluster-wide maximum number of queue groups that can exist at the same time.

- SA_MSG_MAX_NUM_QUEUES_PER_GROUP_ID - This enum can be used to retrieve the maximum number of message queue members in any message queue group.
- SA_MSG_MAX_MESSAGE_SIZE_ID - This enum can be used to retrieve the maximum message size in bytes of a message sent by invoking any of the functions *saMsgMessageSend()*, *saMsgMessageSendAsync()*, or *saMsgMessageSendReceive()*. If this size is exceeded when sending a message, the value SA_AIS_ERR_TOO_BIG is returned by:
  - *saMsgMessageSend()*,
  - *saMsgMessageSendReceive()*, and
  - either *saMsgMessageSendAsync()* or *saMsgMessageDeliveredCallback()*.
- SA_MSG_MAX_REPLY_SIZE_ID - This enum can be used to retrieve the maximum message size in bytes of a reply message sent by invoking any of the functions *saMsgMessageReply()* or *saMsgMessageReplyAsync()*. If this size is exceeded when sending a reply message, the value SA_AIS_ERR_TOO_BIG is returned by:
  - *saMsgMessageReply()* and
  - either *saMsgMessageReplyAsync()* or *saMsgMessageDeliveredCallback()*.

3.5 Library Life Cycle

General remark: If a library call of the Message Service does not complete because the process exited, or because 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.

3.5.1 saMsgInitialize()

Prototype

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

Parameters

msgHandle - [*out*] A pointer to the handle which designates this particular initialization of the Message Service, and which is to be returned by the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

msgCallbacks - [*in*] - If *msgCallbacks* is set to NULL, no callback is registered; If *msgCallbacks* is not set to NULL, it is a pointer to an *SaMsgCallbacksT* structure which contains the callback functions of the process that the Message Service may invoke. Only non-NULL callback functions in this structure will be registered. The *SaMsgCallbacksT* type is defined in Section 3.4.3 on page 24.

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

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

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

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

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

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

See Also

saMsgSelectionObjectGet(), *saMsgDispatch()*, *saMsgFinalize()*

3.5.2 saMsgSelectionObjectGet()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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

Description

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

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

The operating system handle returned by *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).

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

See Also

saMsgInitialize(), *saMsgDispatch()*, *saMsgFinalize()*

3.5.3 saMsgDispatch()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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

Description

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

Return Values

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

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - The *dispatchFlags* parameter is invalid.

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

See Also

saMsgInitialize(), *saMsgSelectionObjectGet()*, *saMsgFinalize()*

3.5.4 saMsgFinalize()

Prototype

SaAisErrorT saMsgFinalize(

SaMsgHandleT msgHandle

);

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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 *saMsgFinalize()* function completes successfully, it releases all resources acquired when *saMsgInitialize()* was called and closes all message queues that are open for the particular handle. Moreover, it stops any tracking associated with the particular handle and frees all resources allocated for it, including the memory allocated for the process in the *saMsgQueueGroupTrack()* function if it has not yet been freed by *saMsgQueueGroupNotificationFree()*. The *saMsgFinalize()* function also frees the memory that is allocated by *saMsgMessageSendReceive()* or *saMsgMessageGet()* if it has not yet been freed by *saMsgMessageDataFree()*. This call also 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.

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

See Also

saMsgInitialize(), *saMsgQueueClose()*, *saMsgQueueGroupTrack()*, *saMsgQueueGroupNotificationFree()*, *saMsgMessageGet()*, *saMsgMessageSendReceive()*, *saMsgMessageDataFree()*, *saMsgSelectionObjectGet()*, *SaMsgQueueOpenCallbackT*, *SaMsgQueueGroupTrackCallbackT*, *SaMsgMessageDeliveredCallbackT*

3.6 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.

3.6.1 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 *saMsgQueueOpenCallback()* 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 by specifying the operating system handle associated with the callback in operating system calls like *poll()* or *select()*. 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.
- **With a blocking call** - The process does not use the operating system handle. It calls instead *saMsgMessageGet()* directly. 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 which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

invocation - [*in*] The invoking process supplies the *invocation* parameter, and the Message Service uses *invocation* when it invokes the corresponding *saMsgQueueOpenCallback()* function to enable the invoking process to associate the callback with the appropriate invocation of *saMsgQueueOpenAsync()*. The *SaInvocationT* type is defined in [1].

queueName - [*in*] A pointer to the name of the message queue to be opened. The *SaNameT* type is defined in [1].

creationAttributes - [*in*] A pointer to the creation attributes of a message queue. The *SaMsgQueueCreationAttributesT* type is defined in Section 3.4.5.2 on page 25.

If the user intends 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 user intends to open a message queue or create and open a message queue if it does not exist, *creationAttributes* must point to a structure containing the attributes for the message queue, and the SA_MSG_QUEUE_CREATE flag in *openFlags* must be set. If the message queue 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. The fields of the *SaMsgQueueCreationAttributesT* structure must be set as follows:

- *creationFlags* - The *creationFlags* field must be set to zero or to SA_MSG_QUEUE_PERSISTENT.
- *size* - The array containing the size in bytes of each priority area of the message queue.
- *retentionTime* - The time duration after a process closed the message queue until it is removed by the Message Service. The *retentionTime* applies only to nonpersistent 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, as defined for the *SaMsgQueueOpenFlagsT* type in Section 3.4.6 on page 25.

timeout - [*in*] The *saMsgQueueOpen()* invocation is considered to have failed if it does not complete within the duration specified. The *SaTimeT* type is defined in [1].

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. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

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 successful 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 *saMsgQueueOpenCallback()* function, which must have been supplied when the process called *saMsgInitialize()*. 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 when invoking *saMsgInitialize()* to initialize the Message Service. These callback functions can be either *saMsgMessageReceivedCallback()*, if the message queue open flag SA_MSG_QUEUE_RECEIVE_CALLBACK is specified, or *saMsgQueueOpenCallback()*. 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, and the name to which *queueName* points 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). In particular, this value is returned if this invocation attempts to create a new message queue, and the maximum number of message queues supported by the implementation is already reached. Refer to the enum SA_MSG_MAX_NUM_QUEUES_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_TOO_BIG - This value is returned if this invocation attempts to create a new message queue, and one or both of the following conditions are met:

- The size of one or more of the priority areas exceeds the maximum size of a priority area supported by the implementation. Refer to the enum SA_MSG_MAX_PRIORITY_AREA_SIZE_ID, defined in Section 3.4.13 on page 34.
- The size of the message queue exceeds the maximum size of a message queue supported by the implementation. Refer to the enum SA_MSG_MAX_QUEUE_SIZE_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_NOT_EXIST - The SA_MSG_QUEUE_CREATE flag is not set, the *creationAttributes* is NULL, and the message queue identified by the name to which *queueName* points does not exist.

SA_AIS_ERR_EXIST - The SA_MSG_QUEUE_CREATE flag is set, the message queue identified by the name to which *queueName* points 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 identified by the name to which *queueName* points is already open.

SA_AIS_ERR_BAD_FLAGS - The *openFlags* parameter or the *creationFlags* field of the structure to which the *creationAttributes* parameter points is invalid.

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

See Also

saMsgQueueClose(), *SaMsgQueueOpenCallbackT*, *SaMsgMessageReceivedCallbackT*, *saMsgQueueRetentionTimeSet()*

3.6.2 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. The *SaInvocationT* type is defined in [1].

queueHandle - [*in*] The handle to the opened message queue. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

error - [*in*] The *error* parameter specifies whether the corresponding invocation of *saMsgQueueOpenAsync()* succeeded or not. The *SaAisErrorT* type is defined in [1]. 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 the name to which *queueName* points 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). In particular, this value is returned if this invocation attempts to create a new message queue, and the maximum number of message queues supported by the implementation is already reached. Refer to the enum SA_MSG_MAX_NUM_QUEUES_ID, defined in Section 3.4.13 on page 34.
- SA_AIS_ERR_TOO_BIG - This value is returned if this invocation attempts to create a new message queue, and one or both of the following conditions are met:
  - The size of one or more of the priority areas exceeds the maximum size of a priority area supported by the implementation. Refer to the enum SA_MSG_MAX_PRIORITY_AREA_SIZE_ID, defined in Section 3.4.13 on page 34.
  - The size of the message queue exceeds the maximum size of a message queue supported by the implementation. Refer to the enum SA_MSG_MAX_QUEUE_SIZE_ID, defined in Section 3.4.13 on page 34.
- SA_AIS_ERR_NOT_EXIST - In the corresponding invocation of the *saMsgQueueOpenAsync()* function, the SA_MSG_QUEUE_CREATE flag is not set, the *creationAttributes* is NULL, and the message queue identified by the name to which *queueName* points does not exist.
- SA_AIS_ERR_EXIST - In the corresponding invocation of the *saMsgQueueOpenAsync()* function, the SA_MSG_QUEUE_CREATE flag is set, the message queue identified by the name to which *queueName* points 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 identified by the name to which *queueName* points 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 *openFlags* parameter or the *creationFlags* field in the structure to which the *creationAttributes* parameter points is invalid.
- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

Description

The 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 calling *saMsgDispatch()* on the handle *msgHandle* that 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()*

3.6.3 saMsgQueueClose()

Prototype

SaAisErrorT saMsgQueueClose(

SaMsgQueueHandleT queueHandle

);

Parameters

queueHandle - [*in*] The handle to the message queue to be closed. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

Description

This API function closes the message queue designated by *queueHandle*. If this call completes successfully, the handle *queueHandle* is no longer valid.

This call frees all resources allocated for this process by the Message Service on the message queue identified by the handle *queueHandle*.

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.

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

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

If the message queue is nonpersistent 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 the retention time elapses, the Message Service deletes the message queue.

The deletion (unlink) of a message queue (see Section 3.6.6) 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 that have this message queue as a member.

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

See Also

saMsgQueueOpen(), *saMsgQueueOpenAsync()*, *SaMsgQueueOpenCallbackT*, *saMsgQueueUnlink()*, *SaMsgMessageReceivedCallbackT*

3.6.4 saMsgQueueStatusGet()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueName - [*in*] A pointer to the name of the message queue whose communication status is to be retrieved. The *SaNameT* type is defined in [1].

queueStatus - [*out*] A pointer to the structure (which is allocated by the invoking process) into which the Message Service writes status information on the message queue identified by the name to which *queueName* points. The *SaMsgQueueStatusT* type is defined in Section 3.4.8.2 on page 27.

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 identified by the name to which *queueName* points cannot be found.

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

See Also

None

3.6.5 saMsgQueueRetentionTimeSet()

Prototype

```
SaAisErrorT saMsgQueueRetentionTimeSet(
        SaMsgQueueHandleT queueHandle,
        SaTimeT *retentionTime
);
```

Parameters

queueHandle - [*in*] The handle to the message queue for which the retention time is to be set. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

retentionTime - [*in*] Pointer to the value of the retention time to be set for the message queue designated by *queueHandle*. The *SaTimeT* type is defined in [1].

Description

The *saMsgQueueRetentionTimeSet()* function sets the retention time of the message queue designated by *queueHandle* to the value to which *retentionTime* points. If the message queue is closed and not reopened by any process within the duration specified by the value to which *retentionTime* points, the Message Service deletes the message queue. The retention time can only be set for nonpersistent 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 with the *saMsgQueueOpen()* or *saMsgQueueOpenCallback()* functions, or the corresponding message queue has already been closed.
- The handle *msgHandle* that was passed to one of 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 because the message queue has been unlinked or is persistent.

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

See Also

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

3.6.6 saMsgQueueUnlink()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueName - [*in*] A pointer to the name of the message queue to be unlinked. The *SaNameT* type is defined in [1].

Description

This function deletes from the cluster the message queue identified by the name to which *queueName* points.

The following applies when a call to this function completes successfully:

- The name to which *queueName* points is no longer valid, that is, any invocation of a function of the Message Service API that uses this message queue name returns an error unless a message queue is re-created with this name. The message queue is re-created by specifying in *saMsgQueueOpen()* or *saMsgQueueOpenAsync()* the SA_MSG_QUEUE_CREATE flag and the same name of the message queue to be unlinked. 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 behavior 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 of which it was a member.

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 identified by the name to which *queueName* points cannot be found.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

See Also

saMsgQueueOpen(), *saMsgQueueOpenAsync()*, *saMsgQueueClose()*

3.7 Management of Message Queue Groups

3.7.1 saMsgQueueGroupCreate()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of the message queue group to be created. The *SaNameT* type is defined in [1].

queueGroupPolicy - [*in*] The message queue group policy. Currently, only the SA_MSG_QUEUE_GROUP_ROUND_ROBIN policy is mandatory. The *SaMsgQueueGroupPolicyT* type is defined in Section 3.4.9 on page 28.

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 the name to which *queueGroupName* points 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). In particular, this value is returned if the maximum number of queue groups supported by the implementation is already reached. Refer to the enum SA_MSG_MAX_NUM_QUEUE_GROUPS_ID, as defined in Section 3.4.13 on page 34.

SA_AIS_ERR_EXIST - The message queue group identified by the name to which *queueGroupName* points already exists.

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

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

See Also

saMsgQueueGroupDelete(), *saMsgQueueGroupInsert()*, *saMsgQueueGroupRemove()*

3.7.2 saMsgQueueGroupInsert()

Prototype

SaAisErrorT saMsgQueueGroupInsert(

SaMsgHandleT msgHandle,

*const SaNameT *queueGroupName,*

*const SaNameT *queueName*

);

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of the message queue group into which the message queue designated by the name to which *queueName* points is to be inserted. The *SaNameT* type is defined in [1].

queueName - [*in*] A pointer to the name of the message queue to be inserted into the message queue group identified by the name to which *queueGroupName* points. The *SaNameT* type is defined in [1].

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). In particular, this value is returned if the maximum number of message queues per message queue group supported by the implementation is already reached. Refer to the enum SA_MSG_MAX_NUM_QUEUES_PER_GROUP_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_NOT_EXIST - The message queue identified by the name to which *queueName* points or the message queue group identified by the name to which *queueGroupName* points cannot be found.

SA_AIS_ERR_EXIST - The message queue identified by the name to which *queueName* points is already a member of the message queue group identified by the name to which *queueGroupName* points.

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

See Also

saMsgQueueGroupRemove(), *saMsgQueueGroupCreate()*, *saMsgQueueGroupDelete()*

3.7.3 saMsgQueueGroupRemove()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of the message queue group from which the message queue identified by the name to which *queueName* points is to be removed. The *SaNameT* type is defined in [1].

queueName - [*in*] A pointer to the name of the message queue to be removed from the message queue group identified by the name to which *queueGroupName* points. The *SaNameT* type is defined in [1].

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 following cases.

- The message queue identified by the name to which *queueName* points or the message queue group designated by the name to which *queueGroupName* points cannot be found.
- The message queue identified by the name to which *queueName* points is not a member of the message queue group designated by the name to which *queueGroupName* points.

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

See Also

saMsgQueueGroupInsert(), *saMsgQueueGroupCreate()*, *saMsgQueueGroupDelete()*

3.7.4 saMsgQueueGroupDelete()

Prototype

SaAisErrorT saMsgQueueGroupDelete(

SaMsgHandleT msgHandle,

*const SaNameT *queueGroupName*

);

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of a message queue group. The *SaNameT* type is defined in [1].

Description

An invocation of this function deletes a message queue group immediately. After this call returns successfully, 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 the name to which *queueGroupName* points cannot be found.

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

See Also

saMsgQueueGroupCreate(), *saMsgQueueGroupInsert()*, *saMsgQueueGroupRemove()*

3.7.5 saMsgQueueGroupTrack()

Prototype

SaAisErrorT saMsgQueueGroupTrack(

SaMsgHandleT msgHandle,

*const SaNameT *queueGroupName,*

SaUint8T trackFlags,

*SaMsgQueueGroupNotificationBufferT *notificationBuffer*

);

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of the message queue group for which tracking of membership is to start. The *SaNameT* type is defined in [1].

trackFlags - [*in*] The kind of tracking that is requested, which is the bitwise OR of one or more of the following flags (as defined in the SA Forum Overview document, see [1]), 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; If *notificationBuffer* is not NULL, this information is returned in the structure to which *notificationBuffer* points 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. The *SaUint8T* type is defined in [1].

notificationBuffer - [*in*/*out*] - A pointer to a buffer of type *SaMsgQueueGroupNotificationBufferT*, as defined in Section 3.4.10.4 on page 30. This parameter is ignored if SA_TRACK_CURRENT is not set in *trackFlags*; if SA_TRACK_CURRENT is set in *trackFlags* and *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 an *in* parameter; otherwise, it specifies that the array to which *notification* points 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, the Message Service has set *numberOfItems* to contain 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 designated by the name to which *queueGroupName* points. The Message Service notifies the process of these changes by invoking the *saMsgQueueGroupTrackCallback()* 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 invocation of *saMsgInitialize()* to initialize the Message Service was incomplete, since the *saMsgQueueGroupTrackCallback()* callback function is missing. This value is not returned if only the SA_TRACK_CURRENT flag is set in *trackFlags* and the *notificationBuffer* is not NULL.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this applies if in the structure to which *notificationBuffer* points, *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 the structure to which *notificationBuffer* points is not NULL, but the *numberOfItems* field in the same structure indicates that the provided buffer is too small to hold information about all members in the message queue group identified by the name to which *queueGroupName* points.

SA_AIS_ERR_NOT_EXIST - The message queue group identified by the name to which *queueGroupName* points cannot be found.

SA_AIS_ERR_BAD_FLAGS - The *trackFlags* parameter is invalid.

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

See Also

saMsgQueueGroupTrackStop(), *SaMsgQueueGroupTrackCallbackT*, *saMsgQueueGroupNotificationFree()*, *saMsgQueueStatusGet()*, *saMsgFinalize()*

3.7.6 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. The *SaNameT* type is defined in [1].

notificationBuffer - [*in*] A pointer to a notification buffer that contains the requested information about the members in the message queue group. The *SaMsgQueueGroupNotificationBufferT* type is defined in Section 3.4.10.4 on page 30.

numberOfMembers - [*in*] The current number of members in the message queue group designated by the name to which *queueGroupName* points. The *SaUint32T* type is defined in [1].

error - [*in*] This parameter indicates whether the Message Service was able to perform the operation. The *SaAisErrorT* type is defined in [1]. 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. In particular, this applies if in the structure to which *notificationBuffer* points, *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. 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 identified by the name to which *queueGroupName* points 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.
- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

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 calling *saMsgDispatch()* on the handle *msgHandle* that was specified when the process invoked the *saMsgQueueGroupTrack()* function to request tracking of membership in a message queue group or changes in the *SaMsgQueueGroupMemberT* structure of any member of the message queue group.

If the *saMsgQueueGroupTrackCallback()* function completes successfully, it returns information about the message queue group members in the structure to which the *notificationBuffer* parameter points. The kind of information returned depends on the setting of the *trackFlags* parameter in the corresponding invocation of the *saMsgQueueGroupTrack()* function.

The value of the *numberOfItems* attribute in the buffer to which the *notificationBuffer* parameter points 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 structure to which *notificationBuffer* points 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()*

3.7.7 saMsgQueueGroupTrackStop()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

queueGroupName - [*in*] A pointer to the name of the message queue group. The *SaNameT* type is defined in [1].

Description

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

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 for one or both of the cases below.

- The message queue group name identified by the name to which *queueGroupName* points cannot be found.
- No track of changes in the membership for the message queue group identified by the name to which *queueGroupName* points was previously started by invoking *saMsgQueueGroupTrack()* with track flags SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY that would still be in effect.

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

See Also

saMsgQueueGroupTrack(), *SaMsgQueueGroupTrackCallbackT*

3.7.8 saMsgQueueGroupNotificationFree()

Prototype

SaAisErrorT saMsgQueueGroupNotificationFree(

SaMsgHandleT msgHandle,

*SaMsgQueueGroupNotificationT *notification*

);

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

notification - [*in*] A pointer to the notification array that was allocated by the Message Service library in the *saMsgQueueGroupTrack()* function and is to be deallocated. The *SaMsgQueueGroupNotificationT* type is defined in Section 3.4.10.3 on page 29.

Description

This function frees the memory to which *notification* points. This memory 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.

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

See Also

saMsgQueueGroupTrack()

3.8 Message Send and Receive Operations

3.8.1 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 which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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. The *SaInvocationT* type is defined in [1].

destination - [*in*] A pointer to the name of a message queue or message queue group to which the message referred to by *message* is sent. The *SaNameT* type is defined in [1].

message - [*in*] A pointer to the message structure specifying the message to be sent. It consists of a buffer, which is provided by the process (pointed to by *data*) and which contains the message data to be sent, a *size* field that contains the size of this 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. The *SaMsgMessageT* type is defined in Section 3.4.11 on page 30.

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 by the *saMsgMessageDeliveredCallback()* function whether the message has been stored in the destination message queue. If the space in the destination message queue is not enough for the entire message, the error SA_AIS_ERR_QUEUE_FULL is returned.
If *ackFlags* is set to zero, the *saMsgMessageDeliveredCallback()* function is not called, meaning that the caller is not informed whether an error occurred or whether it did not.
The *SaMsgAckFlagsT* type is defined in Section 3.4.4 on page 24.

timeout - [*in*] The *saMsgMessageSend()* invocation is considered to have failed if it does not complete within the specified duration. The *SaTimeT* type is defined in [1].

Description

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

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 buffer to which *message->data* points.

The function *saMsgMessageSendAsync()* returns as soon as possible, without waiting for delivery to the destination message queue(s). 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 process must have supplied the *saMsgMessageDeliveredCallback()* when it invoked the *saMsgInitialize()* function.
If *saMsgMessageSendAsync()* returns successfully, and *ackFlags* is not set to zero, the sending process may deallocate the memory for the buffer to which *message->data* points either during an invocation of *saMsgMessageDeliveredCallback()* or after *saMsgMessageDeliveredCallback()* returns.
If *saMsgMessageSendAsync()* returns an error, or *ackFlags* is set to zero (which means that *saMsgMessageDeliveredCallback()* will not be called), the process may deallocate the memory for the buffer to which *message->data* points 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; 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; 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()*; 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. For the message delivery properties, refer to the *saMsgMessageDeliveredCallback()* 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, 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 invocation of *saMsgInitialize()* to initialize the Message Service 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.

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

SA_AIS_ERR_TOO_BIG - This value is returned if the message size of the message to be sent exceeds the maximum message size supported by the implementation. Refer to the enum SA_MSG_MAX_MESSAGE_SIZE_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_NOT_EXIST - The destination message queue name or message queue group name designated by the name to which *destination* points cannot be found.

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

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

SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

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

See Also

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

3.8.2 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. The *SaInvocationT* type is defined in [1].

error - [*in*] This parameter specifies whether the message sent by the corresponding invocation of *saMsgMessageSendAsync()* or *saMsgMessageReplyAsync()* has been delivered to the destination message queue(s) or to the reply buffer supplied by the process that invoked the *saMsgMessageSendReceive()* function. The *SaAisErrorT* type is defined in [1].

- 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_TOO_BIG - This value is returned if
    - the message size of the message to be sent in the corresponding invocation of *saMsgMessageSendAsync()* exceeds the maximum message size supported by the implementation (refer to the enum SA_MSG_MAX_MESSAGE_SIZE_ID, defined in Section 3.4.13 on page 34), or
    - the message size of the reply message in the corresponding invocation of *saMsgMessageReplyAsync()* exceeds the maximum message size of a reply message supported by the implementation. Refer to the enum SA_MSG_MAX_REPLY_SIZE_ID, defined in Section 3.4.13 on page 34.
- SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group designated by the name to which *destination* points 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 to which *senderId* points in the corresponding invocation of *saMsgMessageReplyAsync()* is not large enough to contain the reply message.
- SA_AIS_ERR_QUEUE_FULL - If the name to which *destination* points in the corresponding invocation of the *saMsgMessageSendAsync()* function refers to a message queue, the message queue is full. If the name refers to a unicast message queue group, the selected member message queue is full. If the name refers to 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 points to the name of 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.
- SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.

Description

The 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 calling *saMsgDispatch()* on the handle *msgHandle* that 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

- either the data buffer to which *message->data* points, and which was passed previously to the corresponding *saMsgMessageSendAsync()* function or
- for the data buffer to which *replyMessage->data* points, and which was passed previously to the corresponding *saMsgMessageReplyAsync()* call.

Message delivery properties:

For *saMsgMessageSendAsync()*, the message delivery properties apply to both 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; 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; 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; 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()*

3.8.3 saMsgMessageGet()

Prototype

```
SaAisErrorT saMsgMessageGet(
        SaMsgQueueHandleT queueHandle,
        SaMsgMessageT *message,
        SaTimeT *sendTime,
        SaMsgSenderIdT *senderId,
        SaTimeT timeout
);
```

Parameters

queueHandle - [*in*] The handle of the message queue from which a message is to be received. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

message - [*in*/*out*] A pointer to a message structure of type *SaMsgMessageT* (as defined in Section 3.4.11 on page 30) which contains the following fields:

- *data* - [*in*/*out*] Pointer to a buffer to contain the message data of the message that is to be received.
  If *data* as an *in* parameter is not NULL, it points to memory allocated by the invoking process for the buffer. The size of this memory is given by the *size* parameter.
  If *data* as an *in* parameter is NULL, the value of *size* provided by the invoking process is ignored, and the buffer is provided by the Message Service library. In this case, the buffer must be deallocated by the calling process by invoking the *saMsgMessageDataFree()* function.

- *size* - [*in*/*out*] The size of the buffer to which *data* as an *in* parameter points. If the *saMsgMessageGet()* function returns successfully, *size* contains the actual size of the message data of the received message. If *size* as an *in* parameter is too small, an error is returned, and *size* contains the size required to receive the message.
- *type* - [*out*] The message type of the received message.
- *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 the 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* (as defined in [1]) 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 one of the calls *saMsgMessageSend()*, *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*, as defined in Section 3.4.2 on page 23. If *saMsgMessageGet()* returns SA_AIS_OK and the contents of the field to which *senderId* points is not zero, the receiving thread must reply to the received message by invoking either *saMsgMessageReply()* or *saMsgMessageReplyAsync()* and supplying in either case the same value of **senderId*.

timeout - [*in*] The time duration that specifies how long the *saMsgMessageGet()* function waits for the arrival of a message before it returns with a timeout error.
If *timeout* is set to 0, the call returns immediately; if a message has been found in the message queue before the call returns, and no other error has occurred, this message is returned, and the return value is set to SA_AIS_OK; otherwise, SA_AIS_ERR_TIMEOUT is returned. The *SaTimeT* type is defined in [1].

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 until the time duration specified by the *timeout* parameter elapses, or an error occurs.

The *saMsgMessageGet()* function receives messages in a higher priority area before it receives messages in a lower priority area. Messages with the same priority are received in the order of their arrival in the priority area associated with a priority. For more details, see Section 3.1.5 on page 20.

When the Message Service library allocates the buffer to which *data* points, the invoking process must deallocate this buffer 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 with 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()*.

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

See Also

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

3.8.4 saMsgMessageDataFree()

Prototype

```
SaAisErrorT saMsgMessageDataFree(
        SaMsgHandleT msgHandle,
        void *data
);
```

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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

Description

This function frees the memory to which *data* points. This memory 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.

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

See Also

saMsgMessageGet(), *saMsgMessageSendReceive()*

3.8.5 SaMsgMessageReceivedCallbackT

Prototype

```
typedef void (*SaMsgMessageReceivedCallbackT)(
        SaMsgQueueHandleT queueHandle
);
```

Parameters

queueHandle - [*in*] The handle to the message queue from which the message can be received. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

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 calling *saMsgDispatch()* on the handle *msgHandle* that was specified when the process invoked one of the *saMsgQueueOpen()* or *saMsgQueueOpenAsync()* functions to obtain the handle *queueHandle*.

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

The Message Service invokes this callback 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()*

3.8.6 saMsgMessageCancel()

Prototype

SaAisErrorT saMsgMessageCancel(

SaMsgQueueHandleT queueHandle

);

Parameters

queueHandle - [*in*] The handle to the message queue for which blocking calls to *saMsgMessageGet()* are to be canceled. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

Description

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

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 with 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.

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

See Also

saMsgMessageGet(), *saMsgQueueOpen()*, *saMsgQueueOpenAsync()*

3.9 Request-Reply Operations

3.9.1 saMsgMessageSendReceive()

Prototype

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

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

destination - [*in*] A pointer to the name of a message queue or a unicast message queue group to which the message pointed to by *sendMessage* is to be sent. It is not permitted to specify a multicast message queue group in *destination*. The *SaNameT* type is defined in [1].

sendMessage - [*in*] A pointer to the message structure of *SaMsgMessageT* type (as defined in Section 3.4.11 on page 30) for the message to be sent. The message structure consists 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 that points to the message data, a *size* field that contains the size of the message data, a *priority* field that gives the priority of the message, and a *senderName* pointer.
If *senderName* is not NULL, it points to an area which contains the sender name, and 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 of type *SaMsgMessageT*, as defined in Section 3.4.11 on page 30. This message structure contains the following fields:

- *data* - [*in*/*out*] Pointer to a buffer (the reply buffer) to contain the message data of the reply message.
  If *data* as an *in* parameter is not NULL, it points to memory allocated by the invoking process for the buffer. The size of the buffer is given by the *size* parameter.
  If *data* as an *in* parameter is NULL, the value of *size* provided by the invoking process is ignored, and the reply buffer is provided by the Message Service library. In this case, the reply buffer must be deallocated by the invoking process by calling the *saMsgMessageDataFree()* function.
- *size* - [*in*/*out*] The size of the reply buffer as an *in* parameter. If the *saMsgMessageSendReceive()* function returns successfully, the *size* field as an *out* parameter contains the size of the message data of the reply message. If *size* as an *in* parameter is too small, an error is returned.
- *type* - [*out*] The message type of the received message.
- *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* (as defined in [1]) on successful return from *saMsgMessageSendReceive()*. This value represents the timestamp when the reply message was written into the buffer to which *receiveMessage->data* points. If *replySendTime* as an *in* parameter is NULL, it is ignored.

timeout - [*in*] The time duration that specifies how long the *saMsgMessageSendReceive()* function must wait before it returns with a timeout error. The *SaTimeT* type is defined in [1].

Description

This function enables the transmission of a message to which *sendMessage* points as well as the receipt of a reply message to which *receiveMessage* points in a single invocation. The message to which *sendMessage* points is sent to the message queue or message queue group designated by the name to which *destination* points. The message is sent synchronously, that is, the process blocks waiting for a reply. The reply consists of the message referred to by *receiveMessage*. It must be sent by invoking one of the *saMsgMessageReply()* or *saMsgMessageReplyAsync()* functions, and it must arrive before the time duration specified by the *timeout* parameter elapses; If the reply does not arrive during this time, 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 invocations of *saMsgMessageSendReceive()*. Moreover, the reply message sent by invoking *saMsgMessageReply()* or *saMsgMessageReplyAsync()* is not enqueued in a message queue.

After this call returns, the invoking process may deallocate the memory for the buffer to which *sendMessage->data* points.
When the buffer to which *receiveMessage->data* points is allocated by the Message Service library, the invoking process must deallocate this 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, 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, 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 the *destination* parameter points to the name of a multicast message queue group.

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_TOO_BIG - This value is returned if the message size of the message to be sent exceeds the maximum message size supported by the implementation. Refer to the enum SA_MSG_MAX_MESSAGE_SIZE_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group designated by the name to which *destination* points 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* points to the name of a message queue, the message queue is full. If *destination* points to the name of a unicast message queue group, the selected member message queue is full.

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

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

See Also

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

3.9.2 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 which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

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. The *SaInvocationT* type is defined in [1].

replyMessage - [*in*] A pointer to a structure of type *SaMsgMessageT* (as defined in Section 3.4.11 on page 30) which contains the reply message to be sent. The reply message consists 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 buffer to which *data* points and that contains the message data, a *size* field that contains the size of the message data, a *priority* field that gives the priority of the message, and a *senderName* pointer. 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 which contains the sender name, and

which is supplied by the caller; if *senderName* is NULL, the replying process does not provide the sender name. In this case, if the receiving process (that is the process waiting for a reply in the *saMsgMessageSendReceive()* call) requested a sender name, the *saMsgMessageSendReceive()* call sets the sender name to zero (*receiveMessage->senderName->length* = 0).

senderId - [*in*] Pointer to the same value that the *saMsgMessageGet()* call obtained in **senderId* for the message to which the caller is replying. The *SaMsgSenderIdT* type is defined in Section 3.4.2 on page 23.

ackFlags - [*in*] The kind of the required acknowledgment. This parameter must be set to zero or to SA_MSG_MESSAGE_DELIVERED_ACK. If *ackFlags* is set to zero, the *saMsgMessageDeliveredCallback()* function is not called, and the caller is not informed whether an error occurred or whether it did not. If *ackFlags* is set to SA_MSG_MESSAGE_DELIVERED_ACK, the caller requires to be acknowledged whether the message has been stored in the reply buffer provided in the corresponding *saMsgMessageSendReceive()* call. If the space in the reply buffer is not enough for the entire message, the error SA_AIS_ERR_NO_SPACE is returned. The *SaMsgAckFlagsT* type is defined in Section 3.4.4 on page 24.

timeout - [*in*] The *saMsgMessageReply()* invocation is considered to have failed if it does not complete within the specified duration. The *SaTimeT* type is defined in [1].

Description

These functions are used to reply to a message that was previously sent by calling the *saMsgMessageSendReceive()* function in which case the Message Service sets **senderId* for the message received by the *saMsgMessageGet()* function to a non-zero value. This indicates to the process that it must reply to the received message by invoking either *saMsgMessageReply()* or *saMsgMessageReplyAsync()* and with **senderId* set to the same value that the *saMsgMessageGet()* function set for **senderId*.

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()*, or an error occurs.
After the *saMsgMessageReply()* function returns, the invoking process may deallocate the memory for the buffer to which *replyMessage->data* points.

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 whether an error occurred or whether it did not.
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 invoking process must have supplied the *saMsgMessageDeliveredCallback()* when it called the *saMsgInitialize()* function.
If *saMsgMessageReplyAsync()* returns successfully, and *ackFlags* is not set to zero, the caller may deallocate the memory for the buffer to which *replyMessage->data* points either during the invocation of *saMsgMessageDeliveredCallback()* or after *saMsgMessageDeliveredCallback()* returns.
If *saMsgMessageReplyAsync()* returns an error, or *ackFlags* is set to zero (meaning that *saMsgMessageDeliveredCallback()* will not be called), the caller may deallocate the memory for the buffer to which *replyMessage->data* points 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 the value of the *ackFlags* field is set to SA_MSG_MESSAGE_DELIVERED_ACK, the Message Service will invoke *saMsgMessageDeliveredCallback()*; 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 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 invocation of *saMsgInitialize()* to initialize the Message Service 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 to which this reply is sent was not sent by invoking *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_TOO_BIG - This value is returned if the message size of the reply message exceeds the maximum message size of a reply message supported by the implementation. Refer to the enum SA_MSG_MAX_REPLY_SIZE_ID, defined in Section 3.4.13 on page 34.

SA_AIS_ERR_NOT_EXIST - Either no thread is waiting for a reply, or the value to which the *senderId* parameter points is invalid.

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

SA_AIS_ERR_BAD_FLAGS - The *ackFlags* parameter is invalid.

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

See Also

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

3.10 Set and Get Critical Capacity Thresholds Operations

The section presents one function to set critical capacity thresholds for a message queue and another one to retrieve these thresholds.

3.10.1 saMsgQueueCapacityThresholdsSet()

Prototype

SaAisErrorT saMsgQueueCapacityThresholdsSet(

SaMsgQueueHandleT queueHandle,

*const SaMsgQueueThresholdsT *thresholds*

);

Parameters

queueHandle - [*in*] The handle to the message queue for which the critical capacity thresholds are to be set. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

thresholds - [*in*] Pointer to an *SaMsgQueueThresholdsT* structure (as defined in Section 3.4.12.2 on page 32) that contains the critical capacity thresholds to be set for the message queue identified by the handle *queueHandle*.

Description

This function enables a user application to set the critical capacity thresholds for the priority areas of a message queue. The values set for each priority area *i* must fulfill the following relationships:

0 <= *capacityAvailable*[*i*] <= *capacityReached*[*i*] <= *size*[*i*],

where *size* represents the array containing the sizes of the priority areas as specified at creation time, and *capacityReached* and *capacityAvailable* are arrays in the structure to which *thresholds* points and which contain for each priority area the high-water capacity threshold and the low-water capacity threshold respectively.

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 with 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. This error is returned if one or both of the following conditions are met:

- The *thresholds* pointer is NULL.
- The values set for each priority area *i* do not fulfill the following relationships: 0 <= *capacityAvailable*[*i*] <= *capacityReached*[*i*] <= *size*[*i*], where *capacityAvailable* and *capacityReached* are the arrays in the structure to which *thresholds* points, and *size* represents the array containing the sizes of the priority areas as specified at creation time.

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

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

See Also

saMsgQueueCapacityThresholdsGet(), *saMsgQueueOpen()*, *saMsgQueueOpenAsync()*

3.10.2 saMsgQueueCapacityThresholdsGet()

Prototype

```
SaAisErrorT saMsgQueueCapacityThresholdsGet(
        SaMsgQueueHandleT queueHandle,
        SaMsgQueueThresholdsT *thresholds
);
```

Parameters

queueHandle - [*in*] The handle to the message queue for which the critical capacity thresholds are to be retrieved. The *SaMsgQueueHandleT* type is defined in Section 3.4.1.2 on page 23.

thresholds - [*out*] Pointer to an *SaMsgQueueThresholdsT* structure (as defined in Section 3.4.12.2 on page 32) where the Message Service stores the critical capacity thresholds for the message queue identified by the handle *queueHandle*.

Description

This function enables a user application to retrieve the critical capacity thresholds for 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 *queueHandle* is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with 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. This error is returned if the *thresholds* pointer is NULL.

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

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

See Also

saMsgQueueCapacityThresholdsSet(), *saMsgQueueOpen()*, *saMsgQueueOpenAsync()*

3.11 Retrieve Metadata Size and Limit Fetch Operations

The *saMsgMetadataSizeGet()* and the *saMsgLimitGet()* functions retrieve implementation-specific values.

3.11.1 saMsgMetadataSizeGet()

Prototype

```
SaAisErrorT saMsgMetadataSizeGet(
        SaMsgHandleT msgHandle,
        SaUint32T *metadataSize
);
```

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

metadataSize - [*out*] A pointer to a memory area where the Message Service will return the size of the implementation-specific message metadata portion per message. The *SaUint32T* type is defined in [1].

Description

This function enables a user application to retrieve from the Message Service the size of the implementation-specific message metadata portion.

The message size of a given message *msg* of type *SaMsgMessageT* is calculated as follows:

size of the standard message metadata portion + size of the message data (rounded up to the next multiple of 8 bytes) + *metadataSize*.

The size of the standard message metadata portion is:

```
sizeof (
    struct {
        SaUint32T   type;
        SaUint32T   version;
        SaUint16T   senderNameLength;
        SaUint8T    priority;
        SaSizeT     dataSize;
    }
)
```

plus the size of the sender name, rounded up to the next multiple of 8 bytes.

The fields *type*, *version*, *priority*, and *dataSize* in the preceding structure correspond to the fields *type*, *version*, *priority*, and *size* respectively in the *SaMsgMessageT* structure, as described in Section 3.4.11 on page 30.

The field *senderNameLength* refers to the length field of the *senderName* in the *SaMsgMessageT* structure (that is, *senderName.length*).

The size of the implementation-specific message metadata portion *metadataSize* is not tied to a given message, that is, it is the same for all messages.

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 *metadataSize* pointer is NULL.

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

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

See Also

saMsgInitialize(), *saMsgQueueOpen()*, *saMsgQueueStatusGet()*

3.11.2 saMsgLimitGet()

Prototype

```
SaAisErrorT saMsgLimitGet(
        SaMsgHandleT msgHandle,
        SaMsgLimitIdT limitId,
        SaLimitValueT *limitValue
);
```

Parameters

msgHandle - [*in*] The handle which was obtained by a previous invocation of the *saMsgInitialize()* function and which designates this particular initialization of the Message Service. The *SaMsgHandleT* type is defined in Section 3.4.1.1 on page 23.

limitId - [*in*] The Message Service limit whose implementation-specific value needs to be obtained. The limits are defined in the *SaMsgLimitIdT* type in Section 3.4.13 on page 34.

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

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

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

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

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

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

See Also

saMsgInitialize()

4 Message Service Information Model

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

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

The Message Service UML class diagram has three classes, which show the contained attributes.

4.1 DN Format for the Message Service UML Classes

Table 2 DN Formats for Objects of Message Service Classes

Object Class	DN Format for Objects of the Class
SaMsgQueue	"*safMq=…,** "
SaMsgQueueGroup	"*safMqg=…,** "
SaMsgQueuePriority	"*safMqPrio=…,safMq=…,** "

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

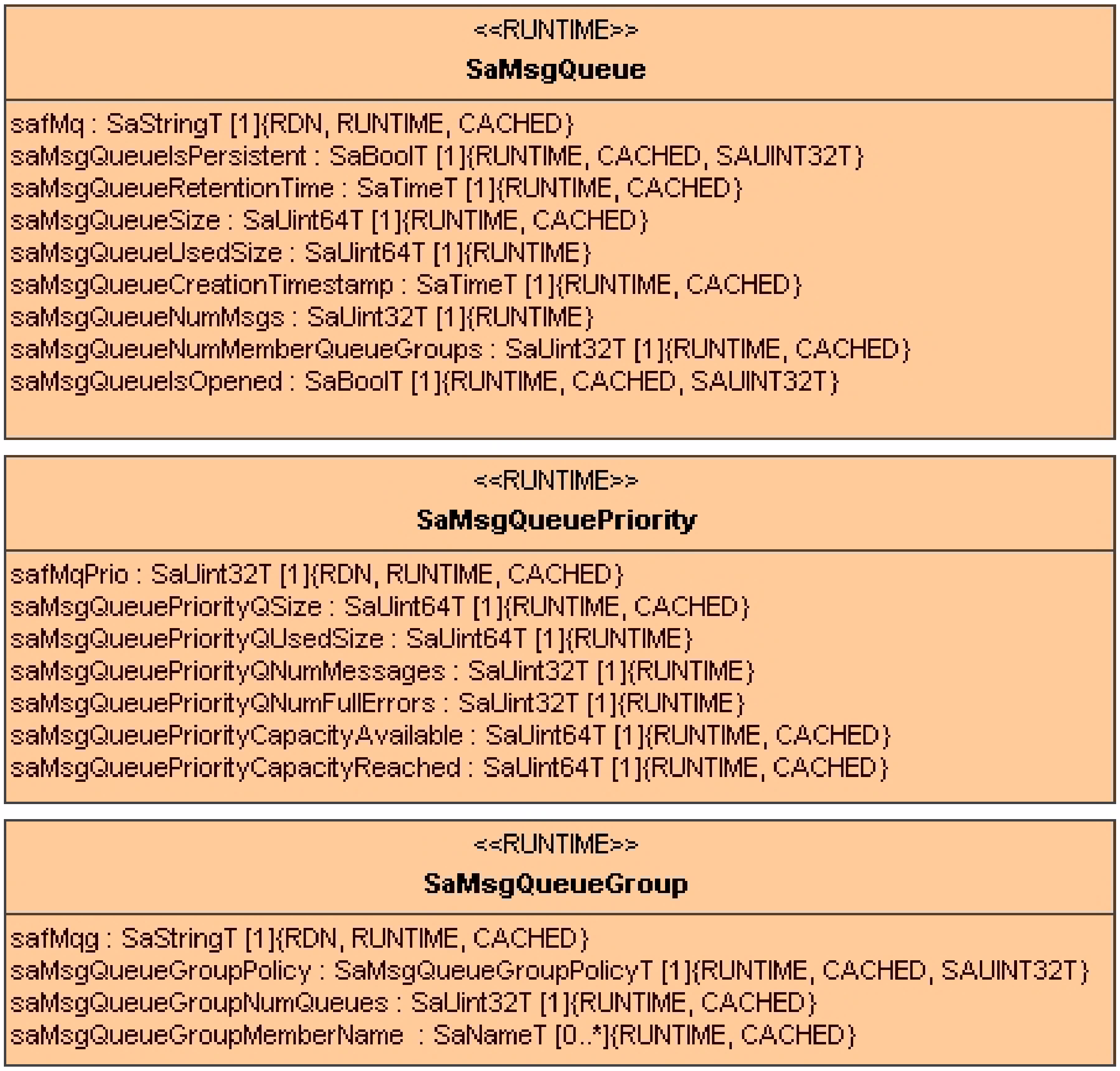
4.2 Message Service UML Classes

The three classes of the Message Service UML model are:

- *SaMsgQueue*—This is a runtime object class that exposes various runtime attributes of a message queue.
- *SaMsgQueuePriority*—This is a runtime object class that exposes various runtime attributes of a message queue priority area.
- *SaMsgQueueGroup*—This is a runtime object class that exposes various runtime attributes of a message queue group.

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

FIGURE 1 Message Service UML Classes


<<RUNTIME>>
SaMsgQueue
safMq : SaStringT [1]{RDN, RUNTIME, CACHED}
saMsgQueueIsPersistent : SaBoolT [1]{RUNTIME, CACHED, SAUINT32T}
saMsgQueueRetentionTime : SaTimeT [1]{RUNTIME, CACHED}
saMsgQueueSize : SaUint64T [1]{RUNTIME, CACHED}
saMsgQueueUsedSize : SaUint64T [1]{RUNTIME}
saMsgQueueCreationTimestamp : SaTimeT [1]{RUNTIME, CACHED}
saMsgQueueNumMsgs : SaUint32T [1]{RUNTIME}
saMsgQueueNumMemberQueueGroups : SaUint32T [1]{RUNTIME, CACHED}
saMsgQueueIsOpened : SaBoolT [1]{RUNTIME, CACHED, SAUINT32T}
<<RUNTIME>>
SaMsgQueuePriority
safMqPrio : SaUint32T [1]{RDN, RUNTIME, CACHED}
saMsgQueuePriorityQSize : SaUint64T [1]{RUNTIME, CACHED}
saMsgQueuePriorityQUsedSize : SaUint64T [1]{RUNTIME}
saMsgQueuePriorityQNumMessages : SaUint32T [1]{RUNTIME}
saMsgQueuePriorityQNumFullErrors : SaUint32T [1]{RUNTIME}
saMsgQueuePriorityCapacityAvailable : SaUint64T [1]{RUNTIME, CACHED}
saMsgQueuePriorityCapacityReached : SaUint64T [1]{RUNTIME, CACHED}
<<RUNTIME>>
SaMsgQueueGroup
safMqg : SaStringT [1]{RDN, RUNTIME, CACHED}
saMsgQueueGroupPolicy : SaMsgQueueGroupPolicyT [1]{RUNTIME, CACHED, SAUINT32T}
saMsgQueueGroupNumQueues : SaUint32T [1]{RUNTIME, CACHED}
saMsgQueueGroupMemberName : SaNameT [0..*]{RUNTIME, CACHED}

5 Message Service Administration API

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

6 Alarms and Notifications

The Message Service produces certain alarms and notifications to convey important information regarding

- its operational and functional state and
- the operational and functional state of the objects under its control

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^{TM} 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 term “Unspecified” for values of attributes. This means that the SA Forum has no specific recommendation on the setting, and the vendor may set these attributes to whatever makes sense in the vendor’s context. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [7]).

6.1 Setting Common Attributes

The tables presented in Section 6.2 refer to attributes defined in [2]. The following list provides recommendations regarding how to populate these attributes.

- Correlation ids - They are supplied to correlate two notifications that have been generated because of a related cause. This attribute is optional; however, in case of alarms that are generated to clear certain conditions, that is, 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 when 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 notification class identifier is an attribute of type *SaNtfClassIdT*. 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 ([1]).

6.2 Message Service Notifications

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

The notifying object must be set to the DN "*safApp=safMsgService*" for the Message Service.

The value of the *majorId* field in the notification class identifier (*SaNtfClassIdT*) must be set to SA_SVC_MSG (as defined in the *SaServicesT* enum in [1]) in all notifications generated by the Message Service.

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)

6.2.1 Message Service Alarms

The Message Service does not issue any alarms at the time of publication of this specification.

6.2.2 Message Service State Change Notifications

6.2.2.1 Message Queue Capacity Reached

Description

All priority areas of the message queue are filled up to their critical capacities (see Section 3.4.12.1 on page 32).

Table 3 Message Queue Capacity Reached Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE
Notification Object	Mandatory	LDAP DN of the message queue (as specified in Section 4.1) that is full and cannot accept any more messages.
Notification Class Identifier	NTF-internal	*minorId* = 0x65
Additional Text	Optional	Unspecified
Additional Information	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

6.2.2.2 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 3.4.12.1 on page 32).

Table 4 Message Queue Capacity Available Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE
Notification Object	Mandatory	LDAP DN of the message queue (as specified in Section 4.1) which is available for receipt of messages.
Notification Class Identifier	NTF-internal	*minorId* = 0x66
Additional Text	Optional	Unspecified
Additional Information	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

6.2.2.3 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 3.4.12.1 on page 32).

Table 5 Message Queue Group Capacity Reached Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE
Notification Object	Mandatory	LDAP DN of the message queue group (as specified in Section 4.1) that is full and cannot accept any more messages.
Notification Class Identifier	NTF-internal	*minorId* = 0x67
Additional Text	Optional	Unspecified
Additional Information	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

6.2.2.4 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 3.4.12.1 on page 32).

Table 6 Message Queue Group Capacity Available Notification

NTF Attribute Name	Attribute Type (NTF-Recommended Value)	SA Forum-Recommended Value
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE
Notification Object	Mandatory	LDAP DN of the message queue group (as specified in Section 4.1) that is available to receive messages.
Notification Class Identifier	NTF-internal	*minorId* = 0x68
Additional Text	Optional	Unspecified
Additional Information	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

7 Message Service Management Interface

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

7.1 Message Service MIB (SAF-MSG-SVC-MIB)

The Message Service MIB contains the three read-only tables *saMsgQueueTable*, *saMsgQueuePriorityTable* and *saMsgQueueGroupTable*, which enumerate the attributes of all currently created message queues, priority areas of a message queue, and message queue groups respectively. The currently created message queues in the cluster include all message queues in the cluster that have not been unlinked as well as the ones that have been unlinked but are still in-use within the cluster.

These tables mimic the UML runtime object classes described in Section 4.2 in terms of the objects contained in the table.

Additionally, the Message Service MIB also defines SNMP traps that correspond to the various notifications for the service defined in Chapter 6 of this specification.

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

Index of Definitions