Service Availability[™] Forum Application Interface Specification

Message Service

SAI-AIS-MSG-B.03.01



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

•



SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT	1
The Service Availability™ Forum Application Interface Specification (the "Package") found at the URL http://www.saforum.org is generally made available by the Service Availability Forum (the "Copyright Holder") for use in developing products that are compatible with the standards provided in the Specification. The terms and conditions which govern the use of the Package are covered by the Artistic License 2.0 of the Perl Foundation, which is reproduced here.	
The Artistic License 2.0	5
Copyright (c) 2000-2006, The Perl Foundation.	
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.	
Preamble	
This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed.	10
The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.	10
You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.	
Definitions	4 5
"Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.	15
"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.	
"You" and "your" means any person who would like to copy, distribute, or modify the Package.	
"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.	20
"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.	
"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.	
"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.	25
"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.	_0
"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.	
"Source" form means the source code, documentation source, and configuration files for the Package.	
"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.	30
Permission for Use and Modification Without Distribution	
(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.	
Permissions for Redistribution of the Standard Version	
(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.	35
(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.	
Distribution of Modified Versions of the Package as Source	
(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:	40

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

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

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

(i) the Original License or

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

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

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

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

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

Aggregating or Linking the Package

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

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

Items That are Not Considered Part of a Modified Version

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

General Provisions

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

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

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

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

(14) Disclaimer of Warranty:

THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. 35

SERVICE

FORUM

1

5

10

15

20



Table of Contents

Message Service ¹

1 Document Introduction	9
1.1 Document Purpose	9 5
1.2 AIS Documents Organization	9
1.3 History	9
1.3.1 New Topics	. 9
1.3.2 Clarifications	. 10
1.3.3 Changes in Return Values of API Functions:	. 11 10
1.3.4 Removed Topics	. 11
1.3.5 Other Changes	. 12
1.4 References	. 12
1.5 How to Provide Feedback on the Specification	. 12
1.6 How to Join the Service Availability TM Forum	13
1.7 Additional Information	13 15
1 7 1 Member Companies	13
1 7 2 Press Materials	13
1.7.2 11055 Waterial5	. 15
2 Overview	. 15
	20
2.1 Message Service	. 15
3 SA Message Service API	. 17
3.1 Message Service Model	. 17
3.1.1 Messages and Message Queues	. 17 25
3.1.2 Message Queue Groups	. 17
3.1.3 Properties of Message Queues	. 19
3.1.3.1 Nonpersistent and Persistent Message Queues	. 19
3.1.3.2 Message Preservation Property of a Queue	. 19
3.1.4 Associating Processes with Message Queues	. 20
3.1.5 Message Delivery Properties	. 20 30
3.2 Unavailability of the Message Service API on a Non-Member Node	. 21
3.2.1 A Member Node Leaves or Rejoins the Cluster Membership	. 21
3.2.2 Guidelines for Message Service Implementers	. 22
3.3 Include File and Library Name	. 23
3.4 Type Definitions	. 23 35
3.4.1 Handles	. 23
3.4.1.1 SaMsgHandleT	. 23
3.4.1.2 SaMsgQueueHandleT	. 23
3.4.2 SaMsgSenderIdT	. 23
3.4.3 SaMsgCallbacksT	. 24 40
3.4.4 SaMsgAckFlagsT	. 24
3.4.5 Message Queue Creation Flags and Creation Attributes	. 24
3.4.5.1 SaMsgQueueCreationFlagsT	. 25



3.4 6 SaMsgUeueOpenFlagsT 25 3.4.7 Message Priority 26 3.4.8 Message Queue Usage and Status 26 3.4.8 1 SaMsQueueEstatusT 27 3.4.9 SaMsQUeueGroupPolicyT 28 3.4.10 1 SaMsQUeueGroupPolicyT 28 3.4.10.1 SaMsQUeueGroupPolicyT 29 3.4.10.3 SaMsQUeueGroupPolicyT 29 3.4.10.3 SaMsQUeueGroupNethertT 29 3.4.10.3 SaMsQUeueGroupNitificationT 29 3.4.11 SaMsQUeueGroupNitificationT 29 3.4.11 SaMsQUeueGroupNitificationT 29 3.4.12 SaMsQUeueGroupNitificationT 30 3.4.12 SaMsQUeueGroupNitificationT 32 3.4.13 SaMsQUeueGroupNitificationT 32 3.4.14 SaMsQUeueGroupNitificationT 32 3.4.12 SaMsQUeueGroupNitificationT 32 3.4.13 SaMsQUeueGroupNitificationT 32 3.4.14 SaMsQUeueGroupNitificationT 32 3.4.15 SamsQUeueGroupNitificationT 34	3.4.5.2 SaMsgQueueCreationAttributesT	25	1
	3.4.6 SaMsgQueueOpenFlagsT	25	
3.4.8 Message Queue Usage and Status 26 3.4.8 LasMsgQueueStatusT 27 3.4.9 SaMsgQueueGroupPolicyT 28 3.4.10 TsoMsgQueueGroupPolicyT 28 3.4.10 TsoMsgQueueGroupPolicyT 29 3.4.10 SaMsgQueueGroupPolicyT 29 3.4.10 SaMsgQueueGroupNotificationT 29 3.4.10 SaMsgQueueGroupNotificationT 29 3.4.11 SaMsgQueueGroupNotificationT 29 3.4.11 SaMsgQueueGroupNotificationT 29 3.4.11 SaMsgQueueGroupNotificationT 29 3.4.12 SaMsgQueueGroupNotificationT 30 3.4.12 SaMsgQueueBrosholdST 32 3.4.12 SaMsgQueueBrosholdST 32 3.4.12 SaMsgQueueBrosholdST 32 3.4.12 SaMsgQueueBrosholdST 32 3.4.13 SaMsgQueueBrosholdST 32 3.5.1 saMsgDispatch() 36 3.5.2 saMsgQueueBrosholdST 36 3.5.3 saMsgQueueDen() and saMsgQueueOpenAsync() 42 3.6 A saMsgQueueDen() and saMsgQueueOpenAsync() 42 3.6.1 saMsgQueueGroupNetter() 52 3.6.3 saMsgQueueGroupPrize() 52 3.6.4 saMsgQueueGroupPrize() 53	3.4.7 Message Priority	26	
$ \begin{array}{ccccccccccccccccccccccccccccccccccc$	3.4.8 Message Queue Usage and Status	26	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.8.1 SaMsgQueueUsageT	27	5
	3.4.8.2 SaMsgQueueStatusT	27	5
	3.4.9 SaMsgQueueGroupPolicyT	28	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.10 Types for Tracking Message Queue Group Changes	28	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.10.1 SaMsgQueueGroupChangesT	28	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.10.2 SaMsgQueueGroupMemberT	29	
3.4.10.4 saMsgQueueToropNotificationBuffer1 30 3.4.11 Critical Capacity of Message Queues and Message Queue Groups 31 3.4.12 SaMsgQueueThresholdsT 32 3.4.12 SaMsgQueueThresholdsT 32 3.4.12 SaMsgQueueThresholdsT 33 3.4.13 SaMsgLimitIdT 34 3.5.1 saMsgInitialize() 36 3.5.1 saMsgDispatch() 36 3.5.2 saMsgSelectionObjectGet() 38 3.5.3 saMsgQueueOpenCallbackT 42 3.6.1 saMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOpenCallbackT 47 3.6.4 saMsgQueueOpenCallbackT 47 3.6.4 saMsgQueueOpenCallbackT 51 3.7.1 saMsgQueueOpenCallbackT 52 3.6.6 saMsgQueueOpenCallbackT 52 3.6.6 saMsgQueueOpenCallbackT 52 3.7.1 saMsgQueueGroupErate() 54 3.7.1 saMsgQueueGroupErate() 56 3.7.1 saMsgQueueGroupDerate() 59 3.7.3 saMsgQueueGroupTrack() 58 3.7.4 saMsgQueueGroupTrack() 58 3.7.5 saMsgQueueGroupTrack() 67 3.7.5 saMsgQueueGroupTrack() 67 3.7.5 saMsgQueueGroupTrac	3.4.10.3 SaMsgQueueGroupNotificationT	29	10
	3.4.10.4 SaMsgQueueGroupNotificationBuffer1	30	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.11 Salvisgiviessage 1	30	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	3.4.12 Childal Capacity of Message Queues and Message Queue Groups		
3.4.12.3 saMsgStateT 33 3.4.13 SaMsgLimitIdT 34 3.5.1 saMsgLimitIdT 36 3.5.1 saMsgInitialize() 36 3.5.1 saMsgDispatch() 40 20 36 3.5.1 saMsgDispatch() 40 3.6 Message Queue Operations 42 3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.5 saMsgQueueClose() 49 25.5 saMsgQueueClose() 49 25.6.1 saMsgQueueClose() 49 3.6.5 saMsgQueueClose() 51 3.6.6 saMsgQueueClose() 54 3.7.1 saMsgQueueGroupInert() 58 3.7.1 saMsgQueueGroupInert() 58 3.7.1 saMsgQueueGroupInert() 58 3.7.1 saMsgQueueGroupTrack(2allbackT 56 3.7.1 saMsgQueueGroupTrack(2allbackT 65 3.7.4 saMsgQueueGroupTrack(2allbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.1 saMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageCloiveredCallbackT 74	3.4.12.1 SalvisgiviessageCapacity status 1	32	
	3 4 12 3 saMsoStateT	33	15
3.5 Library Life Cycle 36 3.5.1 saMsgInitialize() 36 3.5.2 saMsgDispatch() 38 3.5.3 saMsgDispatch() 40 20 3.5.4 saMsgFinalize() 3.6 Message Queue Operations 41 3.6 Message Queue Open() and saMsgQueueOpenAsync() 42 3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueClose() 49 3.6.4 saMsgQueueStatusGet() 51 3.6.5 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueGroupIrackCet() 54 3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupIrack() 58 3.7.3 saMsgQueueGroupIrack() 59 3.7.4 saMsgQueueGroupIrack() 59 3.7.7 saMsgQueueGroupTrackCallbackT 62 3.7.7 saMsgQueueGroupTrackStop() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDeliveredCallbackT <t< td=""><td>3 4 13 SaMsgLimitIdT</td><td>34</td><td></td></t<>	3 4 13 SaMsgLimitIdT	34	
3.5.1 saMsgInitialize() 36 3.5.2 saMsgSelectionObjectGet() 38 3.5.3 saMsgDispatch() 40 20 35.4 saMsgFinilize() 3.6 Message Queue Operations 41 3.6 Message Queue OpenCallbackT 42 3.6.1 saMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOpenCallbackT 47 3.6.5 saMsgQueueClose() 49 3.6.6 saMsgQueueStatusGet() 51 3.6.6 saMsgQueueEntentonTimeSet() 52 3.6.6 saMsgQueueGroupEntet() 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupTrack() 58 3.7.3 saMsgQueueGroupTrack() 58 3.7.4 saMsgQueueGroupTrack() 62 3.7.7 saMsgQueueGroupTrack() 62 3.7.7 saMsgQueueGroupTrack() 67 3.8.1 saMsgUeueGroupTrackStop() 67 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.1 saMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDeliveredCall	3 5 Library Life Cycle	36	
3.5.2 saMsgSelectionObjectGet() 38 3.5.3 saMsgDispatch() 40 20 $3.5.4$ saMsgFinalize() 41 3.6 Message Queue Operations 42 3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOpenCallbackT 47 3.6.4 saMsgQueueOtose() 49 25 3.6.6 saMsgQueueRetentionTimeSet() 3.6.7 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueGroupCreate() 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupRemove() 59 3.7.4 saMsgQueueGroupDelete() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrack(2000) 67 3.7.7 saMsgQueueGroupTrackStop() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.4 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.4 saMsgMessageCancel() 77	3.5.1 saMsoInitialize()	36	
3.5.3 saMsgDispatch() 40 3.5.4 saMsgFinalize() 41 3.6 Message Queue Operations 42 3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOpenCallbackT 47 3.6.4 saMsgQueueClose() 49 3.6.5 saMsgQueueStatusGet() 51 3.6.6 saMsgQueueEtatusGet() 51 3.6.6 saMsgQueueGroupCreate() 54 3.7.1 saMsgQueueGroupCreate() 56 3.7.1 saMsgQueueGroupInsert() 58 3.7.3 saMsgQueueGroupInsert() 58 3.7.4 saMsgQueueGroupTrack() 59 3.7.4 saMsgQueueGroupTrack() 62 3.7.7 saMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 69 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDeliveredCallbackT 74 3.8.5 SaMsgMessageDataFree() 77 3.8.4 saMsgMessageCancel() 77	3.5.2 saMsgSelectionObjectGet()	38	
3.5.4 saMsgFinalize() 41 3.6 Message Queue Operations 42 3.6.1 saMsgQueueOpen() and saMsgQueucOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueClose() 49 3.6.4 saMsgQueueStatusGet() 51 3.6.5 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueBatusGet() 54 3.7.1 saMsgQueueGroupCreate() 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupInsert() 58 3.7.4 saMsgQueueGroupDelete() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 69 3.8 MsgQueueGroupTrackCallbackT 69 3.8 MsgQueueGroupTrackCallbackT 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageCancel() 77 3.8.6 saMsgMessageCancel() 82 </td <td>3.5.2 salvsgDispatch()</td> <td>40</td> <td>20</td>	3.5.2 salvsgDispatch()	40	20
3.6 Message Queue Operations 42 3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOlose() 49 3.6.4 saMsgQueueClose() 51 3.6.5 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueGroupInset() 52 3.6.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupInset() 58 3.7.3 saMsgQueueGroupInset() 58 3.7.4 saMsgQueueGroupInset() 59 3.7.4 saMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 66 3.7.7 saMsgQueueGroupNotificationFree() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.4 saMsgMessageDeliveredCallbackT 74 3.8.5 SaMsgMessageDeliveredCallbackT 80 3.8.5 SaMsgMessageDeliveredCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 5 4 saMsgEinalize()	41	20
3.6.1 saMsgQueueOpen() and saMsgQueueOpenAsync() 42 3.6.2 SaMsgQueueOpenCallbackT 47 3.6.3 saMsgQueueOpenCallbackT 47 3.6.4 saMsgQueueOpenCallbackT 49 3.6.5 saMsgQueueStatusGet() 51 3.6.6 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueGroupInset() 52 3.6.6 saMsgQueueGroupCreate() 56 3.7.1 saMsgQueueGroupInset() 58 3.7.3 saMsgQueueGroupInset() 58 3.7.4 saMsgQueueGroupInset() 59 3.7.4 saMsgQueueGroupInset() 61 3.7.5 saMsgQueueGroupTrack(2) 61 3.7.7 saMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.1 saMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDatFree() 80 3.8.5 SaMsgMessageDatFree() 80 3.8.5 SaMsgMessageDatFree() 80 3.8.6 saMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageConcel() 82	3 6 Message Queue Operations	42	
3.6.2 SaMsgQueuopenCallbackT 47 3.6.3 saMsgQueueClose() 49 3.6.4 saMsgQueueStatusGet() 51 3.6.5 saMsgQueueEtentionTimeSet() 52 3.6.6 saMsgQueueInlink() 54 3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupTreate() 58 3.7.3 saMsgQueueGroupTreate() 59 3.7.4 saMsgQueueGroupTrackCallbackT 61 3.7.5 saMsgQueueGroupTrackCallbackT 62 3.7.7 saMsgQueueGroupTrackStop() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageCet() 77 3.8.3 saMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageCancel() 80 3.8.5 SaMsgMessageCancel() 81 3.8.6 saMsgMessageCancel() 82	3.6.1 saMsgQueueQnen() and saMsgQueueQnenAsync()	42	
3.6.3 saMsgQueueClose() 49 3.6.4 saMsgQueueStatusGet() 51 3.6.5 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueInlink() 54 3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupInsert() 58 3.7.3 saMsgQueueGroupRemove() 59 3.7.4 saMsgQueueGroupTrack() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageGet() 77 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageCancel() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 81	3.6.2 SaMsgQueueOpen() and satisgQueueOpen/ syne()	47	
25.03 and squeuesback() 25 3.6.4 saMsgQueueStatusGet() 51 3.6.5 saMsgQueueRetentionTimeSet() 52 3.6.6 saMsgQueueUnlink() 54 3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupInsert() 58 3.7.3 saMsgQueueGroupRemove() 59 3.7.4 saMsgQueueGroupTrack() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.7.8 saMsgQueueGroupNotificationFree() 69 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageGet() 77 3.8.3 saMsgMessageGet() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3.6.3 saMsgQueueClose()	49	
3.6.5 saMsgQueueBattabet() 51 3.6.5 saMsgQueueUnlink() 52 3.6.6 saMsgQueueUnlink() 54 3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupInsert() 58 3.7.3 saMsgQueueGroupRemove() 59 3.7.4 saMsgQueueGroupDelete() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.7.8 saMsgQueueGroupNtificationFree() 69 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageCancel() 81 3.8.6 saMsgMessageCancel() 81 3.8.6 saMsgMessageCancel() 82	3 6 4 saMsgQueueStatusGet()	51	25
3.6.6 saMsgQueueUnlink()543.7 Management of Message Queue Groups563.7.1 saMsgQueueGroupCreate()563.7.2 saMsgQueueGroupInsert()583.7.3 saMsgQueueGroupRemove()593.7.4 saMsgQueueGroupDelete()613.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupTrackStop()673.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageDeliveredCallbackT743.8.4 saMsgMessageDelatFree()803.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3.6.5 saMsgQueueBetentionTimeSet()	52	
3.7 Management of Message Queue Groups 56 3.7.1 saMsgQueueGroupCreate() 56 3.7.2 saMsgQueueGroupInsert() 58 3.7.3 saMsgQueueGroupRemove() 59 3.7.4 saMsgQueueGroupDelete() 61 3.7.5 saMsgQueueGroupTrack() 62 3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackCallbackT 65 3.7.8 saMsgQueueGroupTrackStop() 67 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageGet() 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageReceivedCallbackT 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 6 6 saMsgQueueUnlink()	54	
3.7.1 saMsgQueueGroupCreate()563.7.2 saMsgQueueGroupInsert()583.7.3 saMsgQueueGroupRemove()593.7.4 saMsgQueueGroupDelete()613.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3.7 Management of Message Queue Groups	56	
3.7.1 saMsgQueueGroupInsert()303.7.2 saMsgQueueGroupInsert()583.7.3 saMsgQueueGroupRemove()593.7.4 saMsgQueueGroupDelete()613.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageDataFree()803.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3.7.1 saMsgOueueGroupCreate()	56	
3.7.2 saMsgQueueGroupRemove()593.7.3 saMsgQueueGroupRemove()613.7.4 saMsgQueueGroupDelete()613.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageReceivedCallbackT813.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3.7.2 saMsgQueueGroupInsert()	58	30
3.7.4 saMsgQueueGroupDelete()613.7.4 saMsgQueueGroupDelete()623.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageGet()773.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3 7 3 saMsgQueueGroupRemove()	59	
3.7.5 saMsgQueueGroupTrack()623.7.6 SaMsgQueueGroupTrackCallbackT653.7.7 saMsgQueueGroupTrackStop()673.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageDataFree()803.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3 7 4 saMsgQueueGroupDelete()	61	
3.7.6 SaMsgQueueGroupTrackCallbackT 65 3.7.7 saMsgQueueGroupTrackStop() 67 3.7.8 saMsgQueueGroupNotificationFree() 69 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 7 5 saMsgQueueGroupTrack()	62	
3.7.7 saMsgQueueGroupTrackStop()67353.7.8 saMsgQueueGroupNotificationFree()693.8 Message Send and Receive Operations703.8.1 saMsgMessageSend() and saMsgMessageSendAsync()703.8.2 SaMsgMessageDeliveredCallbackT743.8.3 saMsgMessageGet()773.8.4 saMsgMessageBotaFree()803.8.5 SaMsgMessageReceivedCallbackT813.8.6 saMsgMessageCancel()82	3 7 6 SaMsgQueueGroupTrackCallbackT	65	
3.7.8 saMsgQueueGroupNotificationFree() 69 3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 7 7 saMsgQueueGroupTrackStop()	67	35
3.8 Message Send and Receive Operations 70 3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 7 8 saMsgOueueGroupNotificationFree()	69	00
3.8.1 saMsgMessageSend() and saMsgMessageSendAsync() 70 3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3.8 Message Send and Receive Operations	70	
3.8.2 SaMsgMessageDeliveredCallbackT 74 3.8.3 saMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3.8.1 saMsoMessageSend() and saMsoMessageSendAsvnc()	70	
3.8.2 SaMsgMessageGet() 77 3.8.4 saMsgMessageDataFree() 80 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3.8.2 SaMsgMessageDeliveredCallbackT	74	
3.8.4 saMsgMessageDataFree() 40 3.8.5 SaMsgMessageReceivedCallbackT 81 3.8.6 saMsgMessageCancel() 82	3 8 3 saMsgMessageGet()	77	
3.8.5 SaMsgMessageReceivedCallbackT	3.8.4 saMsgMessageDataFree()		40
3.8.6 saMsgMessageCancel()	3.8.5 SaMsgMessageReceivedCallbackT		
	3.8.6 saMsgMessageCancel()		



3.9 Request-Reply Operations	1
3.9.1 saMsgMessageSendReceive()	
3.9.2 saMsgMessageReply() and saMsgMessageReplyAsync()	
3.10 Set and Get Critical Capacity Thresholds Operations	
3.10.1 saMsgQueueCapacityThresholdsSet()	5
3.10.2 saMsgOueueCapacityThresholdsGet()	•
3.11 Retrieve Metadata Size and Limit Fetch Operations	
3 11 1 saMs@MetadataSizeGet() 95	
3 11 2 saMsgLimitGet() 97	
	10
4 Message Service Information Model	10
4.1 DN Format for the Message Service UML Classes 99	
4.7 Massage Sarvice UML Classes 00	
4.2 Micssage Scivice UNIL Classes	
5 Message Service Administration API 101	15
6 Alarms and Notifications	
6.1 Setting Common Attributes	
6.2 Message Service Notifications 104	
6.2.1 Message Service Alarms 104	20
6.2.2 Message Service State Change Notifications 105	20
6.2.2 Message Oueue Canacity Reached	
6.2.2.2 Message Queue Capacity Available 106	
6.2.2.3 Message Queue Group Canacity Reached 107	
6.2.2.4 Message Queue Group Capacity Available	
	25
7 Message Service Management Interface 109	
7.1 Message Service MIB (SAF-MSG-SVC-MIB) 109	



5
10
15
20
25
30
35



1	Document Introduction	1
1.1	Document Purpose	
	This document defines the Message Service of the Application Interface Specification (AIS) of the Service Availability [™] Forum (SA Forum). It is intended for use by implementers of the Application Interface Specification and by application developers who would use the Application Interface Specification to develop applications that must be highly available. The AIS is defined in the C programming language, and requires substantial knowledge of the C programming language.	5 10
	Typically, the Service Availability [™] Forum Application Interface Specification will be used in conjunction with the Service Availability [™] Forum Hardware Interface Specification (HPI).	15
1.2	AIS Documents Organization	
	The Application Interface Specification is organized into several volumes. For a list of all Application Interface Specification documents, refer to the SA Forum Overview document ([1]).	20
1.3	History	
	Previous releases of the Message Service specification:	05
	(1) SAI-AIS-MSG-A.01.01	25
	(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.	30
1.3	.1 New Topics	
	 Section 3.1.1 introduces the notion of message metadata. This implies clarifica- tions in Section 3.4.5.2 and Section 3.4.8.1. 	35
	 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. 	40



•	Section 3.4.13 describes the <i>SaMsgLimitIdT</i> enum, which provides a set of values that identify limits for a particular implementation of the Message Service.	1
•	Section 3.10 describes the set and get capacity thresholds operations.	
•	Section 3.11 describes functions to retrieve implementation-specific values (mes- sage metadata size and a particular implementation limit).	5
•	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.	10
•	Chapter 5 states that no administration APIs are provided for the Message Ser- vice.	
•	Chapter 7 presents the Message Service Management Interface.	
1.3.2 Clar	ifications	15
•	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 <i>saMsgDispatch()</i> function clarifies the meaning of the SA_AIS_OK return value.	20
•	The description of the <i>saMsgFinalize()</i> 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.	25
•	The description of the <i>saMsgQueueClose()</i> function (see Section 3.6.3) clarifies which resources this function frees for the invoking process.	20
•	The return values subsection of Section 3.7.5 on the <i>saMsgQueueGroupTrack()</i> function clarifies when the SA_AIS_ERR_INIT is returned.	• •
•	Section 6.2 clarifies the setting of the notifying object in notifications.	30



1.3.3 Changes in Return Values of AF	PI Functions:		1
Table 1 Changes in R	eturn Values of API Functions		_
API Function	Return Value	Change Type	5
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	10
saMsgQueueOpen(), saMsgQueueOpenAsync(), and SaMsgQueueOpenCallbackT	SA_AIS_ERR_NO_RESOURCES	extended	20
SaMsgQueueGroupTrackCallbackT	SA_AIS_ERR_INVALID_PARAM	clarified]
saMsgMessageSend() and saMsgMessageSendAsync()	SA_AIS_ERR_NO_MEMORY SA_AIS_ERR_NO_RESOURCES	extended	25

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" 35

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	1
 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 con- figuration is provided. 	5
 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 Ser- vice. 	5
1.4 References	10
The following documents contain information that is relevant to the specification:	
[1] Service Availability [™] Forum, Service Availability Interface, Overview, SAI-Overview-B.03.01	15
[2] Service Availability [™] Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.02.01	
[3] Service Availability [™] Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.01.01	20
[4] Service Availability [™] Forum, Application Interface Specification, Cluster Mem- bership Service, SAI-AIS-CLM-B.03.01	
[5] Service Availability [™] Forum, SA Forum Information Model in XML Metadata Interchange (XMI) v2.1 format, SAI-XMI-A.02.01	05
[6] Service Availability [™] Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.02.01	25
[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.	30
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 (<u>http://www.saforum.org</u>).	35

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



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



5
10
15
20
25
30
35



2	Overview
	This specification defines the Message Service within the Application Interface Spec- ification (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 pro- cess 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 distrib- uted, 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 mes- sage 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 a service to 1

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.

⁴⁰

^{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]).



5
10
15
20
25
30
35





3 SA Message Service API	1
3.1 Message Service Model	
3.1.1 Messages and Message Queues	5
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.	10
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.	15
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.	20
A message queue is global to a cluster and is identified by a unique name in the name space of all message queues.	25
Message queues can be dynamically created and deleted.	
3.1.2 Message Queue Groups	30
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.	35
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.	40
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	



5

15

20

25

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 10 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.	1
3.1.3 Properties of Message Queues	5
3.1.3.1 Nonpersistent and Persistent Message Queues	0
A message queue can be defined as nonpersistent , meaning that the Message Ser- vice 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 mes- sage queue is closed.	10
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 <i>saMsgQueueUnlink()</i> function.	15
If no process has the message queue opened when <i>saMsgQueueUnlink()</i> 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.	20
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.	25
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.	30 35
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.	40



3.1.4	Associating Processes with Message Queues	1
	Messages are sent and received by processes. The Message Service is a coopera- tive model where any process may write to any message queue or message queue group.	5
	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.	10
	If a process terminates abnormally, the Message Service automatically closes all of its open message queues.	
3.1.5	Message Delivery Properties	15
	• Priority - When a process receives a message from a message queue by invoking <i>saMsgMessageGet()</i> , 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.	20
	 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 mes- sage queues. 	25 30
	 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 	35
	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.	40

If a process sends a message to a multicast message queue group and



requests an acknowledgment, SA_AIS_OK is retu sent successfully to at least one member of the m	Irned if the message can be essage queue group.	1
 Acknowledgment - A process sending messages Service to notify the process whether the sending can ask for an acknowledgment that the sent mess destination message queue (refer to send operati the reply to a sending process has been received (refer to request/reply operations in Section 3.9). 	s can request the Message was successful. A process sage has been stored in the ons in Section 3.8.1) or that by the sending process	5
 Persistence of messages - Messages are kept i sage never expires in a message queue. When a cessfully from a message queue by invoking saM message is removed from the message queue. Th a message queue may reside on disk, on a cluster 	n message queues. A mes- message is retrieved suc- sgMessageGet(), the ne physical representation of r file system, on global	D
shared memory, on shared memory of each node and so on. However, the choice of persistence ca the performance of the Message Service. Therefor not require that the physical representation of a m stored to survive node failures or shutting down th See also Section 3.1.3.2.	with or without replication, 18 n have negative effects on ore, this specification does lessage queue be durably he entire cluster.	5
2.0 Unavailability of the Massage Comise ADI on a	Zu Nara Marahar Nada	J
3.2 Unavailability of the Message Service API on a	Non-Wember Node	
The Message Service <u>does not</u> provide service to proces are not in the cluster membership (see [4]).	sses on cluster nodes that 2	5
The following subsection describes the behavior of the Mous conditions that cause the Message Service to be una 3.2.2 contains recommendations to Message Service imporary unavailability of providing service.	lessage Service under vari- available on a node. Section blementers for dealing with a 30	0
3.2.1 A Member Node Leaves or Rejoins the Cluster Member	ership	
If the cluster node has left the cluster membership (see [tively evicted from the cluster membership, the Message towards processes residing on that node and using or at	4]) or is being administra- Service behaves as follows tempting to use the service: 35	5
⇒ Calls to <i>saMsgInitialize()</i> will fail with SA_AIS_ERR_	UNAVAILABLE.	

 \Rightarrow 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: 40

 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. 	1
 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; oth- erwise, SaMsgMessageDeliveredCallbackT will not be called. 	10
 The saMsgFinalize() function, which is used to free the library handles and all resources associated with these handles. 	15
⇒ Any outstanding callbacks SaMsgQueueOpenCallbackT, SaMsgQueueGroupTrackCallbackT, and SaMsgMessageDeliveredCallbackT will return SA_AIS_ERR_UNAVAILABLE in the error parameter.	
\Rightarrow The callback SaMsgMessageReceivedCallbackT will not be called.	20
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 <i>saMsgFinalize()</i> , 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.	25 30
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 <i>saMsgQueueUnlink()</i> operation is pending because one or more processes on the leaving node had the message queue open, the unlink operation can proceed now.	35

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.	1
3.3 Include File and Library Name	5
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></samsg.h>	10
To use the Message Service API, an application must be bound with the following library:	
libSaMsg.so	15
3.4 Type Definitions	
The Message Service uses the types described in the following sections.	
3.4.1 Handles	20
3.4.1.1 SaMsgHandleT	
typedef SaUint64T SaMsgHandleT;	25
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.	20
3.4.1.2 SaMsgQueueHandleT	30
typedef SaUint64T SaMsgQueueHandleT;	
This type is used for the handle to a message queue.	
3.4.2 SaMsgSenderIdT	35
typedef SaUint64T SaMsgSenderIdT;	
This type is used internally by the Message Service to identify the thread that called <i>saMsgMessageSendReceive()</i> ; it must not be changed by the invoking thread.	40



3.4.3 SaMsgCallbacksT	1
The SaMsgCallbacksT structure is defined as follows:	
typedef struct { SaMsgQueueOpenCallbackT saMsgQueueOpenCallback;	5
SaMsgQueueGroupTrackCallbackT saMsgQueueGroupTrackCallback; SaMsgMessageDeliveredCallbackT saMsgMessageDeliveredCallback; SaMsgMessageReceivedCallbackT saMsgMessageReceivedCallback; } SaMsgCallbacksT;	10
A structure of the <i>SaMsgCallbacksT</i> type (called a callbacks structure) is used to specify the callback functions that the Message Service may invoke.	15
3.4.4 SaMsgAckFlagsT	
The SaMsgAckFlagsT type is used in the saMsgMessageSendAsync() and saMsgMessageReplyAsync() calls. A parameter of the type SaMsgAckFlagsT indi- cates the kind of the required acknowledgment and can be set to either zero or SA_MSG_MESSAGE_DELIVERED_ACK:	20
#define SA_MSG_MESSAGE_DELIVERED_ACK 0x1	
<i>typedef SaUint32T SaMsgAckFlagsT;</i> SA_MSG_MESSAGE_DELIVERED_ACK - This flag indicates that the caller requires	25
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.	30
3.4.5 Message Queue Creation Flags and Creation Attributes	35
This section defines the creation flags and the creation attributes of a message queue used in the <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> calls.	



3.4.5.1 SaMsgQueueCreationFlagsT	1
#define SA_MSG_QUEUE_PERSISTENT 0x1	
typedef SaUint32T SaMsgQueueCreationFlagsT;	Б
SA_MSG_QUEUE_PERSISTENT - If this flag is set, the message queue is persis- tent, that is, it can be removed only by an explicit call to <i>saMsgQueueUnlink()</i> . If the SA_MSG_QUEUE_PERSISTENT flag is not set, the message queue is nonpersis- tent, 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 reten- tion time of the message queue elapses.	10
3.4.5.2 SaMsgQueueCreationAttributesT	
typedef struct {	15
SaMsgQueueCreationFlagsT creationFlags;	
SaSizeT size[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];	
SaTimeT retentionTime;	
} SaMsgQueueCreationAttributesT;	20
The fields of the SaMsgQueueCreationAttributesT type have the following interpreta- tion:	
 creationFlags - Zero or SA_MSG_QUEUE_PERSISTENT. Refer also to retentionTime below. 	25
 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 meta- data 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). 	30
 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. 	35
3.4.6 SaMsgQueueOpenFlagsT	
The following values specify the open attributes used in the <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> calls.	40



#	#define SA_MSG_QUEUE_CREATE	0x1	1
#	#define SA_MSG_QUEUE_RECEIVE_CALLBACK	0x2	
#	#define SA_MSG_QUEUE_EMPTY	0x4	
t	ypedef SaUint32T SaMsgQueueOpenFlagsT;		5
Д r	A value of the <i>SaMsgQueueOpenFlagsT</i> type is zero on the flags in the following list:	or the bitwise OR of one or	
	 SA_MSG_QUEUE_CREATE - This flag reques ate a message queue if the message queue do 	ts the Message Service to cre- es not exist.	10
	 SA_MSG_QUEUE_RECEIVE_CALLBACK - The Service to notify the process about the arrival of saMsgMessageReceivedCallback() callback can retrieved by invoking saMsgMessageGet(). If the function for the arrival of messages is not called invoke the saMsgMessageGet() call to receive 	his flag requests the Message of messages by invoking the all. The message can be his flag is not set, the callback d, but the process can still messages.	15
	 SA_MSG_QUEUE_EMPTY - This flag requests existing messages when opening a message q Message Service must preserve existing messa sage queue. However, the Message Service ca will be preserved in a fail-over situation. 	the Message Service to delete ueue. If the flag is not set, the ages when opening the mes- nnot guarantee that messages	20
3.4.7 M	essage Priority		
N a	Messages can have a priority between SA_MSG_MES and SA_MSG_MESSAGE_HIGHEST_PRIORITY.	SSAGE_LOWEST_PRIORITY	25
ŧ	#define SA_MSG_MESSAGE_HIGHEST_PRIORITY	0	
#	#define SA_MSG_MESSAGE_LOWEST_PRIORITY	3	30
3.4.8 M	essage Queue Usage and Status		
T v	This section defines the types <i>SaMsgQueueUsageT</i> a which are used to obtain information about a message	and SaMsgQueueStatusT, e queue.	35



3.4.8.1 SaMsgQueueUsageT	1
typedef struct {	
SaSizeT queueSize;	5
SaSizeT queueUsed;	Ū
SaUint32T numberOfMessages;	
} SaMsgQueueUsageT;	
The fields of the <i>SaMsgQueueUsageT</i> type are valid for a given priority area of a message queue. They have the following interpretation:	10
 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 meta- data. 	15
 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. 	00
 numberOfMessages - The current number of messages in the priority area. 	20
3.4.8.2 SaMsgQueueStatusT	
typedef struct {	25
SaMsgQueueCreationFlagsT creationFlags;	20
SaTimeT retentionTime;	
SaTimeT closeTime;	
SaMsgQueueUsageT saMsgQueueUsage[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];	30
} SaMsgQueueStatusT;	
The fields of the SaMsgQueueStatusT type have the following interpretation:	35
 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. 	40



 <i>closeTime</i> - The absolute time when the message queue was last closed. This field contains zero if the message queue is currently open for a process. <i>saMsgQueueUsage</i> - An array containing the message queue usage data for each priority area of the message queue. 	1
3.4.9 SaMsgQueueGroupPolicyT	0
typedef enum {	
$SA_MSG_QUEUE_GROUP_ROUND_ROBIN = 1,$	10
SA_MSG_QUEUE_GROUP_LOCAL_ROUND_ROBIN = 2,	-
SA_MSG_QUEUE_GROUP_LOCAL_BEST_QUEUE = 3,	
SA_MSG_QUEUE_GROUP_BROADCAST = 4	
} SaMsgQueueGroupPolicyT;	15
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.	20
3.4.10 Types for Tracking Message Queue Group Changes	20
This section defines the types needed for tracking message queue group changes.	
3.4.10.1 SaMsgQueueGroupChangesT	25
typedef enum {	20
SA_MSG_QUEUE_GROUP_NO_CHANGE = 1,	
$SA_MSG_QUEUE_GROUP_ADDED = 2,$	
$SA_MSG_QUEUE_GROUP_REMOVED = 3,$	30
SA_MSG_QUEUE_GROUP_STATE_CHANGED = 4	
} SaMsgQueueGroupChangesT;	
The values of the SaMsgQueueGroupChangesT enumeration type have the following interpretation:	35
 SA_MSG_QUEUE_GROUP_NO_CHANGE - This value is used when the <i>trackFlags</i> parameter of the <i>saMsgQueueGroupTrack()</i> function (as defined in Section 3.7.5) is either SA_TRACK_CURRENT or 	40
 SA_TRACK_CHANGES and 	



3.4.

3.4.

 the message queue was already a member of the message queue group in the previous saMsgQueueGroupTrackCallback() callback call, 	1
 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. 	5
 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. 	10
10.2 SaMsgQueueGroupMemberT	
typedef struct {	15
SaNameT queueName;	IJ
} SaMsgQueueGroupMemberT;	
The field <i>queueName</i> is the name of a message queue in the message queue group.	
10.3 SaMsgQueueGroupNotificationT	20
typedef struct {	
SaMsgQueueGroupMemberT member;	
SaMsgQueueGroupChangesT change;	25
} SaMsgQueueGroupNotificationT;	
The fields of the <i>SaMsgQueueGroupNotificationT</i> type have the following interpreta- tion:	
 member - Information about a message queue group member. 	30
 change - The type of change since the last callback. 	
	35



3.4.10.4 SaMsgQueueGroupNotificationBufferT				
typedef struct {				
SaUint32T numberOfItems;	5			
SaMsgQueueGroupNotificationT *notification;	0			
SaMsgQueueGroupPolicyT queueGroupPolicy;				
} SaMsgQueueGroupNotificationBufferT;				
The fields of the SaMsgQueueGroupNotificationBufferT type have the following inter- pretation:	10			
 <i>numberOfItems</i> - Number of elements in the array to which <i>notification</i> points. Each element in this array is of type <i>SaMsgQueueGroupNotificationT</i>. <i>notification</i> - Pointer to the notification array 	15			
 <i>queueGroupPolicy</i> - The load distribution policy of the message queue group. 				
3 / 11 SaMegMoesagoT				
This two describes a message to be used for conding and receiving				
This type describes a message to be used for sending and receiving.				
typedef struct {				
SaUint32T type;				
SaUint32T version;	25			
SaSizeT size;				
SaNameT *senderName;				
void *data;				
SaUint8T priority;	30			
} SaMsgMessageT;				
The fields of the SaMsgMessageT type have the following interpretation:				
 <i>type</i> - Message type that is specified by a process when it sends a message. <i>version</i> - 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 <i>version</i> and to ensure that the application program can handle messages with appropriate different versions. 	35			
 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. 	40			



senderName - A pointer to a name that identifies the sender of the message. It 1 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 5 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. 10 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 15 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 20 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 intro-25 duced. 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-30 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. 35 The user can set the critical capacity of the priority areas of a queue by invoking the saMsqQueueCapacityThresholdsSet() 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. 40

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

SERVICE AVAILABILITY FORUM

3.4.12.1 saMsgMessageCapacityStatusT	1
The following enumeration contain values representing the various states of a mes- sage queue or a message queue group that need to be notified to a system adminis- trator in form of a state change notification.	5
typedef enum {	
SA_MSG_QUEUE_CAPACITY_REACHED = 1,	
$SA_MSG_QUEUE_CAPACITY_AVAILABLE = 2,$	10
$SA_MSG_QUEUE_GROUP_CAPACITY_REACHED = 3,$	10
SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE = 4	
} SaMsgMessageCapacityStatusT;	
The values of the SaMsgMessageCapacityStatusT enumeration type have the follow- ing interpretation:	15
 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. 	20
 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. 	25
 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. 	30
3.4.12.2 SaMsgQueueThresholdsT	35
typedef struct {	
SaSizeT capacityReached[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];	
SaSizeT capacityAvailable[SA_MSG_MESSAGE_LOWEST_PRIORITY+1];	40
} SaMsgQueueThresholdsT;	τU



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>i</i> must fulfill the fol-
lowing 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;

1

5

10

15

25

30

35

3.4.13 SaMsgL	imitIdT		1
The Salv impleme does not impleme	<i>IsgLimitIdT</i> enum provides a set of values that iden ntation of the Message Service. Note that the Message define a configuration for these limits, which are us ntation.	tify limits for a given age Service specification sually predefined by the	5
The user the ident below) w	can retrieve at runtime the current value of a partic ifier of the limit (one of the enum values of the type hen invoking the <i>saMsgLimitGet()</i> function (see Se	cular limit by specifying <i>SaMsgLimitIdT</i> , defined ction 3.11.2 on page 97).	10
The limit defined i <i>uint64Va</i>	value is returned in a parameter of a generic type (n [1]). As all limits defined in this specification are o <i>lue</i> field of <i>SaLimitValueT</i> must be used for further	<i>SaLimitValueT</i> type, f type <i>SaUint64T</i> , the access.	10
typedef e	enum {		15
S	A_MSG_MAX_PRIORITY_AREA_SIZE_ID	= 1,	10
S	A_MSG_MAX_QUEUE_SIZE_ID	= 2,	
S	A_MSG_MAX_NUM_QUEUES_ID	= 3,	
S	A_MSG_MAX_NUM_QUEUE_GROUPS_ID	= 4,	20
S	A_MSG_MAX_NUM_QUEUES_PER_GROUP_ID	= 5,	
S	A_MSG_MAX_MESSAGE_SIZE_ID	= 6,	
S	A_MSG_MAX_REPLY_SIZE_ID	= 7	25
} SaMsg	LimitIdT;		
The valu	es of the SaMsgLimitIdT enumeration type have the	e 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 		30	
valu by S	e can be smaller, equal, or larger than the humber of A_MSG_MAX_PRIORITY_AREA_SIZE_ID.	or priority areas multiplied	35

- 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. 40



 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. 	1
 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 func- tions saMsgMessageSend(), saMsgMessageSendAsync(), or saMsgMessageSendReceive(). If this size is exceeded when sending a mes- sage, the value SA_AIS_ERR_TOO_BIG is returned by: 	5
 saMsgMessageSend(), 	10
 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: 	15
 saMsgMessageReply() and 	
 either saMsgMessageReplyAsync() or saMsgMessageDeliveredCallback(). 	20

3.5	Library Life Cycle	1
	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.	5
3.5.	.1 saMsgInitialize()	
	Prototype	10
	SaAisErrorT saMsgInitialize(
	SaMsgHandleT *msgHandle,	
	const SaMsgCallbacksT *msgCallbacks,	15
	SaVersionT *version	15
);	
	Parameters	20
	<i>msgHandle</i> - [<i>out</i>] 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 <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	20
	<i>msgCallbacks</i> - [<i>in</i>] - If <i>msgCallbacks</i> is set to NULL, no callback is registered; If <i>msgCallbacks</i> is not set to NULL, it is a pointer to an <i>SaMsgCallbacksT</i> 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 <i>SaMsgCallbacksT</i> type is defined in Section 3.4.3 on page 24.	25
	<i>version</i> - [<i>in</i> / <i>out</i>] As an input parameter, <i>version</i> is a pointer to a structure containing the required Message Service version. In this case, <i>minorVersion</i> is ignored and should be set to 0x00	30
	As an output parameter, <i>version</i> is a pointer to a structure containing the version actually supported by the Message Service. The <i>SaVersionT</i> type is defined in [1].	35
	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 <i>msgHandle</i> is returned by the Message Service as the reference to this association between the	40


process and the Message Service. The process uses this handle in subsequent com- munication with the Message Service.	1
If the implementation supports the specified <i>releaseCode</i> and <i>majorVersion</i> , SA_AIS_OK is returned. In this case, the structure pointed to by the <i>version</i> parame- ter is set by this function to:	5
 releaseCode = required release code 	
 majorVersion = highest value of the major version that this implementation can support for the required releaseCode 	10
 minorVersion = highest value of the minor version that this implementation can support for the required value of releaseCode and the returned value of majorVersion 	10
If the preceding condition cannot be met, SA_AIS_ERR_VERSION is returned, and the structure pointed to by the <i>version</i> parameter is set to:	15
if (implementation supports the required releaseCode)	
releaseCode = required releaseCode	
else {	20
if (implementation supports <i>releaseCode</i> higher than the required <i>releaseCode</i>)	20
<i>releaseCode</i> = the lowest value of the supported release codes that is higher than the required <i>releaseCode</i>	25
else	20
<i>releaseCode</i> = the highest value of the supported release codes that is lower than the required <i>releaseCode</i>	
}	30
<i>majorVersion</i> = highest value of the major versions that this implementation can sup- port for the returned <i>releaseCode</i>	
<i>minorVersion</i> = highest value of the minor versions that this implementation can sup- port for the returned values of <i>releaseCode</i> and <i>majorVersion</i>	35
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.	40



	SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	1
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	5
	SA_AIS_ERR_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.	10
	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 <i>version</i> parameter points is not compatible with the version of the Message Service implementation.	15
	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	
	See Also	20
	saMsgSelectionObjectGet(),	
3.5.2	saMsgSelectionObjectGet()	
	Prototype	25
	SaAisErrorT saMsgSelectionObjectGet(
	SaMsgHandleT msgHandle,	
	SaSelectionObjectT *selectionObject	30
);	
	Parameters	
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the	35

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].

Message Service. The SaMsgHandleT type is defined in Section 3.4.1.1 on page 23.



Description	1
This function returns the operating system handle associated with the handle <i>msgHandle</i> . The invoking process can use the operating system handle to detect pending callbacks, instead of repeatedly invoking the <i>saMsgDispatch()</i> function for this purpose.	5
In a POSIX environment, the operating system handle is a file descriptor that is used with the <i>poll()</i> or <i>select()</i> system calls to detect incoming callbacks.	
The operating system handle returned by <i>saMsgSelectionObjectGet()</i> is valid until <i>saMsgFinalize()</i> is invoked on the same handle <i>msgHandle</i> .	10
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.	15
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	20
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	25
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).	30
SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	05
See Also	35
saMsgInitialize(), saMsgDispatch(), saMsgFinalize()	

3.5.3	saMsgDispatch()	1
	Prototype	
	SaAisErrorT saMsgDispatch(5
	SaMsgHandleT msgHandle,	
	SaDispatchFlagsT dispatchFlags	
);	10
	Parameters	10
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	15
	<i>dispatchFlags</i> - [<i>in</i>] Flags that specify the callback execution behavior of the <i>saMsgDispatch()</i> function, which have the values SA_DISPATCH_ONE, SA_DISPATCH_ALL, or SA_DISPATCH_BLOCKING. These flags are values of the <i>SaDispatchFlagsT</i> enumeration type, which is described in [1].	20
	Description	
	In the context of the calling thread, this function invokes pending callbacks for the handle <i>msgHandle</i> in the way specified by the <i>dispatchFlags</i> parameter.	25
	Return Values	
	SA_AIS_OK - The function completed successfully. This value is also returned if this function is being invoked with <i>dispatchFlags</i> set to SA_DISPATCH_ALL or SA_DISPATCH_BLOCKING, and the handle <i>msgHandle</i> has been finalized.	30
	SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	
	SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	35
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	40
	SA_AIS_ERR_INVALID_PARAM - The dispatchFlags parameter is invalid.	



	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	1
	See Also	
	saMsgInitialize(), saMsgSelectionObjectGet(), saMsgFinalize()	5
3.5.4	saMsgFinalize()	
	Prototype	10
	SaAisErrorT saMsgFinalize(
	SaMsgHandleT msgHandle	
);	15
	Parameters	15
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	20
	Description	
	The <i>saMsgFinalize()</i> function closes the association represented by the <i>msgHandle</i> parameter between the invoking process and the Message Service. The process must have invoked <i>saMsgInitialize()</i> before it invokes this function. A process must invoke this function once for each handle it acquired by invoking <i>saMsgInitialize()</i> .	25
	If the <i>saMsgFinalize()</i> function completes successfully, it releases all resources acquired when <i>saMsgInitialize()</i> 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 <i>saMsgQueueGroupTrack()</i> function if it has not yet been freed by <i>saMsgQueueGroupNotificationFree()</i> . The <i>saMsgFinalize()</i> function also	30
	frees the memory that is allocated by <i>saMsgMessageSendReceive()</i> or <i>saMsgMessageGet()</i> if it has not yet been freed by <i>saMsgMessageDataFree()</i> . 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.	35
	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.	40



	After <i>saMsgFinalize()</i> completes successfully, the handle <i>msgHandle</i> and the selec- tion object associated with it are no longer valid.	1
	Return Values	
	SA_AIS_OK - The function completed successfully.	5
	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.	10
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	15
	See Also	
	saMsgInitialize(), saMsgQueueClose(), saMsgQueueGroupTrack(), saMsgQueueGroupNotificationFree(), saMsgMessageGet(), saMsgMessageSendReceive(), saMsgMessageDataFree(), saMsgSelectionObjectGet(), SaMsgQueueOpenCallbackT, SaMsgQueueGroupTrackCallbackT, SaMsgMessageDeliveredCallbackT	20
		25
3.6 N	lessage Queue Operations	
	In the following description, when it is said that a process is receiving from the <i>destination</i> message queue, it also includes the case of a process receiving from a message queue that is a member of a <i>destination</i> message queue group.	30
3.6.1	saMsgQueueOpen() and saMsgQueueOpenAsync()	
	The <i>saMsgQueueOpen()</i> and <i>saMsgQueueOpenAsync()</i> functions create and open a new message queue or open an existing message queue. The <i>saMsgQueueOpen()</i> function is a synchronous blocking operation that returns a new message queue handle. The <i>saMsgQueueOpenAsync()</i> function is an asynchronous operation; the corresponding <i>saMsgQueueOpenCallback()</i> returns the new message queue handle to the invoking process.	35
		40

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 <i>poll()</i> or <i>select()</i>. After being notified, the process can invoke <i>saMsgDispatch()</i>, which in turn calls <i>saMsgMessageReceivedCallback()</i>. In the callback, or after its completion, the 	1 5
process can invoke <i>saMsgMessageGet()</i> 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 success- fully if a message can be retrieved from the message queue within a specified time limit. 	10
Prototype	
SaAisErrorT saMsgQueueOpen(15
SaMsgHandleT msgHandle,	15
const SaNameT *queueName,	
const SaMsgQueueCreationAttributesT *creationAttributes,	
SaMsgQueueOpenFlagsT openFlags,	20
SaTimeT timeout,	
SaMsgQueueHandleT *queueHandle	
);	25
SaAisErrorT saMsgQueueOpenAsync(20
SaMsgHandleT msgHandle,	
SalnvocationT invocation,	
const SaNameT *queueName,	30
const SaMsgQueueCreationAttributesT *creationAttributes,	
SaMsgQueueOpenFlagsT openFlags	
);	35
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.



<i>invocation</i> - [<i>in</i>] The invoking process supplies the <i>invocation</i> parameter, and the Message Service uses <i>invocation</i> when it invokes the corresponding <i>saMsgQueueOpenCallback()</i> function to enable the invoking process to associate the callback with the appropriate invocation of <i>saMsgQueueOpenAsync()</i> . The <i>SaInvocationT</i> type is defined in [1].	1 5
<i>queueName</i> - [<i>in</i>] A pointer to the name of the message queue to be opened. The <i>SaNameT</i> type is defined in [1].	
<i>creationAttributes</i> - [<i>in</i>] A pointer to the creation attributes of a message queue. The <i>SaMsgQueueCreationAttributesT</i> type is defined in Section 3.4.5.2 on page 25.	10
If the user intends only to open an existing message queue, <i>creationAttributes</i> must be set to NULL and the SA_MSG_QUEUE_CREATE flag in <i>openFlags</i> must not be set.	15
If the user intends to open a message queue or create and open a message queue if it does not exist, <i>creationAttributes</i> must point to a structure containing the attributes for the message queue, and the SA_MSG_QUEUE_CREATE flag in <i>openFlags</i> 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 <i>creationAttributes->retentionTime</i> , which is ignored, as it may be independently mod- ified by invoking the <i>saMsgQueueRetentionTimeSet()</i> function. The fields of the <i>SaMsgQueueCreationAttributesT</i> structure must be set as follows:	20
 creationFlags - The creationFlags field must be set to zero or to SA_MSG_QUEUE_PERSISTENT. 	25
 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. 	30
<i>openFlags</i> - [<i>in</i>] 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 <i>SaMsgQueueOpenFlagsT</i> type in Section 3.4.6 on page 25.	35
<i>timeout</i> - [<i>in</i>] The <i>saMsgQueueOpen()</i> invocation is considered to have failed if it does not complete within the duration specified. The <i>SaTimeT</i> type is defined in [1].	40
<i>queueHandle</i> - [<i>out</i>] 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 <i>saMsgQueueOpen()</i> function. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	1
Description	
The <i>saMsgQueueOpen()</i> and <i>saMsgQueueOpenAsync()</i> functions open a message queue. If the message queue does not exist, and the SA_MSG_QUEUE_CREATE flag is set in the <i>openFlags</i> parameter, the message queue is created first.	5
After successful completion of the invocation of <i>saMsgQueueOpen()</i> , which is a blocking call, the Message Service returns a message queue handle to the message queue in the <i>queueHandle</i> parameter.	10
For <i>saMsgQueueOpenAsync(</i>), the Message Service returns a message queue han- dle when it invokes the <i>saMsgQueueOpenCallback(</i>) function, which must have been supplied when the process called <i>saMsgInitialize(</i>). The process invoking <i>saMsgQueueOpenAsync(</i>) sets the <i>invocation</i> parameter and the Message Service uses it in the corresponding callback call.	15
Both for the <i>saMsgQueueOpen()</i> and <i>saMsgQueueOpenAsync()</i> functions, if the message queue open flag SA_MSG_QUEUE_RECEIVE_CALLBACK is specified, the <i>saMsgMessageReceivedCallback()</i> callback function must have been supplied when invoking <i>saMsgInitialize()</i> previously.	20
The open operation is needed to receive messages from the message queue.	
Return Values	25
SA_AIS_OK - The function completed successfully.	20
SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred, or the timeout defined by the <i>timeout</i> parameter occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	30
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	35
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	00
SA_AIS_ERR_INIT - One or more callback functions were not supplied when invok- ing <i>saMsgInitialize()</i> to initialize the Message Service. These callback functions can be either <i>saMsgMessageReceivedCallback()</i> , if the message queue open flag SA_MSG_QUEUE_RECEIVE_CALLBACK is specified, or	40



saMsgQueueOpenCallback(). The latter case only applies to saMsgQueueOpenAsync().	1
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this value is returned if one of the cases below apply:	5
 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. 	10
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.	15
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:	20
 The size of one or more of the priority areas exceeds the maximum size of a pri- ority 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. 	25
 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 <i>creationAttributes</i> is NULL, and the message queue identified by the name to which <i>queueName</i> points does not exist.	30
SA_AIS_ERR_EXIST - The SA_MSG_QUEUE_CREATE flag is set, the message queue identified by the name to which <i>queueName</i> points exists, and one or both of the values <i>creationAttributes->creationFlags</i> or <i>creationAttributes->size</i> are different from the corresponding values used at creation time.	35
SA_AIS_ERR_BUSY - The message queue identified by the name to which <i>queueName</i> points is already open.	40
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.	1
See Also	
saMsgQueueClose(), SaMsgQueueOpenCallbackT, SaMsgMessageReceivedCallbackT, saMsgQueueRetentionTimeSet()	5
3.6.2 SaMsgQueueOpenCallbackT	
Prototype	10
typedef void(*SaMsgQueueOpenCallbackT)(
SaInvocationT invocation,	
SaMsgQueueHandleT queueHandle,	15
SaAisErrorT error	
);	
Parameters	20
<i>invocation</i> - [<i>in</i>] A designator that associates this invocation to a previous call to the <i>saMsgQueueOpenAsync()</i> function. The <i>SaInvocationT</i> type is defined in [1].	
<i>queueHandle</i> - [<i>in</i>] The handle to the opened message queue. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	25
<i>error</i> - [<i>in</i>] The <i>error</i> parameter specifies whether the corresponding invocation of <i>saMsgQueueOpenAsync()</i> succeeded or not. The <i>SaAisErrorT</i> type is defined in [1]. The possible values of the <i>error</i> parameter are:	30
 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. 	35
 SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later. 	40
 SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> in the corresponding invocation of the <i>saMsgQueueOpenAsync()</i> function is invalid, since it is cor- rupted, uninitialized, or has already been finalized. 	40



 SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly in the corresponding invocation of the saMsgQueueOpenAsync() function. In particula this value is returned if one of the cases below apply: 	- 1 ⁻ ,
 The SA_MSG_QUEUE_CREATE flag is not set, and creationAttributes is not NULL. 	5
 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. 	o 10
 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 queue 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. 	5 15 S
 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: 	20 20
 The size of one or more of the priority areas exceeds the maximum size of 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. 	a 25
 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. 	20
 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. 	30
 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->creationElags or 	35
creationAttributes->size are different from the corresponding values used at creation time.	40



 SA_AIS_ERR_BUSY - The message queue identified by the queueName points in the corresponding invocation of the saMsgQueueOpenAsync() function is already open. 	ne name to which 1
 SA_AIS_ERR_BAD_FLAGS - In the corresponding invoca saMsgQueueOpenAsync() function, the openFlags parame creationFlags field in the structure to which the creationAtt points is invalid. 	tion of the 5 eter or the 5 <i>ributes</i> parameter
 SA_AIS_ERR_UNAVAILABLE - The operation requested in able on this cluster node because it is not a member node 	n this call is unavail- 10
Description	
The Message Service invokes this callback function when the ope the invocation of <i>saMsgQueueOpenAsync()</i> completes. This calls the context of a thread calling <i>saMsgDispatch()</i> on the handle <i>ms</i> specified in the <i>saMsgQueueOpenAsync()</i> call.	ration requested by back is invoked in <i>gHandle</i> that was 15
The reference to the opened/created message queue is returned only if <i>error</i> is SA_AIS_OK; If the call is not successful, an error is <i>error</i> parameter.	in <i>queueHandle</i> returned in the 20
Return Values	
None	
See Also	25
saMsgQueueOpenAsync(),	0
3.6.3 saMsgQueueClose()	
Prototype	30
SaAisErrorT saMsgQueueClose(
SaMsgQueueHandleT queueHandle	
);	35
Parameters	
<i>queueHandle</i> - [<i>in</i>] The handle to the message queue to be close <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page	d. The 23. 40



5

10

15

20

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. 25

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:

40

35



	 It is corrupted, was not obtained with the saMsgQueueOpen() or saMsgQueueOpenCallback() functions, or the corresponding message queue has already been closed. 	1
	 The handle msgHandle that was passed to the functions saMsgQueueOpen() or saMsgQueueOpenAsync() has already been finalized. 	5
	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	10
	saMsgQueueOpen(), saMsgQueueOpenAsync(), SaMsgQueueOpenCallbackT, saMsgQueueUnlink(), SaMsgMessageReceivedCallbackT	
3.6.4	saMsgQueueStatusGet()	45
	Prototype	15
	SaAisErrorT saMsgQueueStatusGet(
	SaMsgHandleT msgHandle,	20
	const SaNameT *queueName,	20
	SaMsgQueueStatusT *queueStatus	
);	
	Parameters	25
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	30
	<i>queueName</i> - [<i>in</i>] A pointer to the name of the message queue whose communication status is to be retrieved. The <i>SaNameT</i> type is defined in [1].	
	<i>queueStatus</i> - [<i>out</i>] 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 <i>queueName</i> points. The <i>SaMsgQueueStatusT</i> type is defined in Section 3.4.8.2 on page 27.	35
	Description	40
	This function retrieves information about the status of a message queue.	



5

15

20

Return Values

SA_AIS_OK - The function completed successfully.

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

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- 10 cess 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	25
None	

3.6.5 saMsgQueueRetentionTimeSet()

Prototype	30		
SaAisErrorT saMsgQueueRetentionTimeSet(
SaMsgQueueHandleT queueHandle,			
SaTimeT *retentionTime	35		
);			

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.



<i>retentionTime</i> - [<i>in</i>] Pointer to the value of the retention time to be set for the message queue designated by <i>queueHandle</i> . The <i>SaTimeT</i> type is defined in [1].	1
Description	
The <i>saMsgQueueRetentionTimeSet()</i> function sets the retention time of the message queue designated by <i>queueHandle</i> to the value to which <i>retentionTime</i> points. If the message queue is closed and not reopened by any process within the duration specified by the value to which <i>retentionTime</i> points, the Message Service deletes the message queue. The retention time can only be set for nonpersistent message queues.	5 10
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.	15
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	20
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> is invalid, due to one or both of the reasons below:	25
 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. 	30
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	
SA_AIS_ERR_BAD_OPERATION - The retention time of the message queue desig- nated by <i>queueHandle</i> cannot be changed because the message queue has been unlinked or is persistent.	35
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	40
saMsgQueueOpen(), saMsgQueueOpenAsync(), saMsgQueueClose(), saMsgQueueOpenCallbackT, saMsgQueueUnlink()	

saMsgQueueUnlink()	
Prototype	
SaAisErrorT saMsgQueueUnlink(
SaMsgHandleT msgHandle,	
const SaNameT *queueName	
);	
Parameters	
<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page	23.
<i>queueName</i> - [<i>in</i>] A pointer to the name of the message queue to be unlinked. Th <i>SaNameT</i> type is defined in [1].	е
Description	
This function deletes from the cluster the message queue identified by the name to which <i>queueName</i> points.	to
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 poly not yet finally deleted. 	on ie nes- ne ossi-
 Note that this behavior is similar to the way POSIX treats files. If no process has the message queue open when <i>saMsgQueueUnlink()</i> is invoked, the message queue is immediately deleted even if its creation attrib is SA MSG QUEUE PERSISTENT. 	ute
 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 	d.
The deletion of a message queue frees all resources allocated by the Message S vice for it.	er-



When <i>saMsgQueueUnlink()</i> has successfully completed, the message queue has been removed from all message queue groups of which it was a member.	1
This API can be invoked by any process and the invoking process need not be the creator or opener of the message queue.	5
Return Values	
SA_AIS_OK - The function completed successfully.	
SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	10
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	. –
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	15
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	
SA_AIS_ERR_NOT_EXIST - The message queue identified by the name to which <i>queueName</i> points cannot be found.	20
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.	25
See Also	
saMsgQueueOpen(),	30

40



3.7 Management of Message Queue Groups	1
3.7.1 saMsgQueueGroupCreate()	
Prototype	5
SaAisErrorT saMsgQueueGroupCreate(
SaMsgHandleT msgHandle,	
const SaNameT *queueGroupName,	10
SaMsgQueueGroupPolicyT queueGroupPolicy	
);	
Parameters	15
<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	
<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group to be cre- ated. The <i>SaNameT</i> type is defined in [1].	20
<i>queueGroupPolicy</i> - [<i>in</i>] The message queue group policy. Currently, only the SA_MSG_QUEUE_GROUP_ROUND_ROBIN policy is mandatory. The <i>SaMsgQueueGroupPolicyT</i> type is defined in Section 3.4.9 on page 28.	25
Description	
This function creates a message queue group of a particular policy. The current ver- sion of the specification only requires round robin load distribution to be supported.	30
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.	35
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.	40
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	10



SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	1
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this value is returned if the name to which <i>queueGroupName</i> points is not a DN, or the type of its first RDN is not <i>safMqg</i> .	5
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.	10
SA_AIS_ERR_EXIST - The message queue group identified by the name to which <i>queueGroupName</i> points already exists.	15
SA_AIS_ERR_NOT_SUPPORTED - The specified <i>queueGroupPolicy</i> is not sup- ported 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.	20
See Also	
saMsgQueueGroupDelete(),	25

3.7.2 saMsgQueueGroupInsert()		
	Prototype	
	SaAisErrorT saMsgQueueGroupInsert(5
	SaMsgHandleT msgHandle,	
	const SaNameT *queueGroupName,	
	const SaNameT *queueName	10
);	10
	Parameters	
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	15
	<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group into which the message queue designated by the name to which <i>queueName</i> points is to be inserted. The <i>SaNameT</i> type is defined in [1].	20
	<i>queueName</i> - [<i>in</i>] A pointer to the name of the message queue to be inserted into the message queue group identified by the name to which <i>queueGroupName</i> points. The <i>SaNameT</i> type is defined in [1].	25
	Description	
	This function inserts a message queue into a message queue group.	
	Return Values	30
	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.	35
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	40
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	



S	SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	1
s ti	SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the provider of he service is out of memory and cannot provide the service.	
s n q r d	SA_AIS_ERR_NO_RESOURCES - There are insufficient resources (other than memory). In particular, this value is returned if the maximum number of 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.	5
S Q Q	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.	10
s q tl	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 he name to which queueGroupName points.	15
s ti	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on his cluster node because it is not a member node.	
S	See Also	20
S S	saMsgQueueGroupRemove(), saMsgQueueGroupCreate(), saMsgQueueGroupDelete()	
3.7.3 sa	aMsgQueueGroupRemove()	25
F	Prototype	
S	SaAisErrorT saMsgQueueGroupRemove(
	SaMsgHandleT msgHandle,	30
	const SaNameT *queueGroupName,	
	const SaNameT *queueName	
)	· •	35
F	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.

<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group from which the message queue identified by the name to which <i>queueName</i> points is to be removed. The <i>SaNameT</i> type is defined in [1].	1 ?
<i>queueName</i> - [<i>in</i>] A pointer to the name of the message queue to be removed from the message queue group identified by the name to which <i>queueGroupName</i> points The <i>SaNameT</i> type is defined in [1].	
Description	
This function removes a message queue from a message queue group.	10
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.	15
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it dic not.	l 20
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	25
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. 	30
 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. 	35
SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable or this cluster node because it is not a member node.	۱
See Also	
saMsgQueueGroupInsert(),	40



3.7.4 s	3.7.4 saMsgQueueGroupDelete()	
	Prototype	
	SaAisErrorT saMsgQueueGroupDelete(5
	SaMsgHandleT msgHandle,	
	const SaNameT *queueGroupName	
);	10
	Parameters	10
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	15
	<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of a message queue group. The SaNameT type is defined in [1].	
	Description	20
	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	25
	SA_AIS_OK - The function completed successfully.	
	SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	30
	SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	35
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	
	SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	40
	SA_AIS_ERR_NOT_EXIST - The message queue group identified by the name to which <i>queueGroupName</i> 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.	1
See Also	
saMsgQueueGroupCreate(),	5
3.7.5 saMsgQueueGroupTrack()	
Prototype	10
SaAisErrorT saMsgQueueGroupTrack(
SaMsgHandleT msgHandle,	
const SaNameT *queueGroupName,	15
SaUint8T trackFlags,	
SaMsgQueueGroupNotificationBufferT *notificationBuffer	
);	20
Parameters	
<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	25
<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group for which tracking of membership is to start. The <i>SaNameT</i> type is defined in [1].	
<i>trackFlags</i> - [<i>in</i>] 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:	30
 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 com- pletes successfully. 	35
 SA_TRACK_CHANGES - The notification callback is invoked each time a change occurs in the membership of the message queue group. The callback 	40



call provides an <i>SaMsgQueueGroupNotificationT</i> structure for <u>all</u> members (changed and not changed) of the message queue group.	1
 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. 	5
It is not permitted to set both SA_TRACK_CHANGES and SA_TRACK_CHANGES_ONLY in an invocation of this function. The <i>SaUint8T</i> type is defined in [1].	10
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 <i>trackFlags</i> ; if SA_TRACK_CURRENT is set in <i>trackFlags</i> and <i>notificationBuffer</i> 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:	15
 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. 	20 25
 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. 	30
Description	
This function starts tracking changes in the membership of a message queue group designated by the name to which <i>queueGroupName</i> points. The Message Service notifies the process of these changes by invoking the <i>saMsgQueueGroupTrackCallback()</i> callback function, which must have been supplied when the process invoked the <i>saMsgInitialize()</i> call.	35
An application may call <i>saMsgQueueGroupTrack()</i> repeatedly for the same values of <i>msgHandle</i> and <i>queueGroupName</i> , regardless of whether the call initiates a one-time status request or a series of callback notifications. If <i>saMsgQueueGroupTrack()</i> is	40

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 <i>msgHandle</i> and <i>queueGroupName</i> , the Message Service will invoke further notification callbacks according to SA_TRACK_CHANGES_ONLY. The same is true vice versa. Once <i>saMsgQueueGroupTrack()</i> has been called with <i>trackFlags</i> containing either SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, notification callbacks can only be stopped by an invocation of the <i>saMsgQueueGroupTrackStop()</i> function.	1 5
Return Values	
SA_AIS_OK - The function completed successfully.	10
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.	15
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	20
SA_AIS_ERR_INIT - The previous invocation of <i>saMsgInitialize()</i> to initialize the Mes- sage Service was incomplete, since the <i>saMsgQueueGroupTrackCallback()</i> callback function is missing. This value is not returned if only the SA_TRACK_CURRENT flag is set in <i>trackFlags</i> and the <i>notificationBuffer</i> is not NULL.	25
SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this applies if in the structure to which <i>notificationBuffer</i> points, <i>notification</i> is not NULL, and <i>numberOfItems</i> is 0.	30
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).	35
SA_AIS_ERR_NO_SPACE - The SA_TRACK_CURRENT flag is set, and the <i>notification</i> field in the structure to which <i>notificationBuffer</i> points is not NULL, but the <i>numberOfItems</i> 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 <i>queueGroupName</i> points.	40
SA_AIS_ERR_NOT_EXIST - The message queue group identified by the name to which <i>queueGroupName</i> points cannot be found.	



	SA_AIS_ERR_BAD_FLAGS - The trackFlags parameter is invalid.	1
	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	5
	saMsgQueueGroupTrackStop(),	
3.7.6	SaMsgQueueGroupTrackCallbackT	10
	Prototype	
	typedef void (*SaMsgQueueGroupTrackCallbackT) (
	const SaNameT *queueGroupName,	15
	const SaMsgQueueGroupNotificationBufferT *notificationBuffer,	
	SaUint32T numberOfMembers,	
	SaAisErrorT error	20
);	20
	Parameters	
	<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group. The <i>SaNameT</i> type is defined in [1].	25
	<i>notificationBuffer</i> - [<i>in</i>] A pointer to a notification buffer that contains the requested information about the members in the message queue group. The <i>SaMsgQueueGroupNotificationBufferT</i> type is defined in Section 3.4.10.4 on page 30.	30
	<i>numberOfMembers</i> - [<i>in</i>] The current number of members in the message queue group designated by the name to which <i>queueGroupName</i> points. The <i>SaUint32T</i> type is defined in [1].	35
	<i>error</i> - [<i>in</i>] This parameter indicates whether the Message Service was able to per- form the operation. The <i>SaAisErrorT</i> type is defined in [1]. The parameter <i>error</i> has one of the values:	
	 SA_AIS_OK - The function completed successfully. 	40
	 SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore. 	



5

10

20

25

30

35

•	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 40 queue group or changes in the *SaMsgQueueGroupMemberT* structure of any member of the message queue group.



	If the <i>saMsgQueueGroupTrackCallback()</i> function completes successfully, it returns information about the message queue group members in the structure to which the <i>notificationBuffer</i> parameter points. The kind of information returned depends on the setting of the <i>trackFlags</i> parameter in the corresponding invocation of the <i>saMsgQueueGroupTrack()</i> function.	1 5
	The value of the <i>numberOfItems</i> attribute in the buffer to which the <i>notificationBuffer</i> parameter points might be greater than the value of the <i>numberOfMembers</i> 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 <i>notificationBuffer</i> 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.	10
	If an error occurs, it is returned in the error parameter.	15
	Return Values	
	None	
	See Also	20
	saMsgQueueGroupTrack(),	
3.7.7 s	saMsgQueueGroupTrackStop()	25
	Prototype	
	SaAisErrorT saMsgQueueGroupTrackStop(
	SaMsgHandleT msgHandle,	30
	const SaNameT *queueGroupName	
);	
	Parameters	35
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	40
	<i>queueGroupName</i> - [<i>in</i>] A pointer to the name of the message queue group. The <i>SaNameT</i> type is defined in [1].	40

5

10

20

25

30

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. 15

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 ³⁵ this cluster node because it is not a member node.

See Also

saMsgQueueGroupTrack(), SaMsgQueueGroupTrackCallbackT 40



3.7.8 saN	MsgQueueGroupNotificationFree()	1
Pr	rototype	
Sa	aAisErrorT saMsgQueueGroupNotificationFree(5
	SaMsgHandleT msgHandle,	
	SaMsgQueueGroupNotificationT *notification	
);		10
Ра	arameters	
ms sa Me	sgHandle - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>MsgInitialize()</i> function and which designates this particular initialization of the essage Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	15
no Se Th	<i>otification</i> - [<i>in</i>] A pointer to the notification array that was allocated by the Message ervice library in the <i>saMsgQueueGroupTrack()</i> function and is to be deallocated. The <i>SaMsgQueueGroupNotificationT</i> type is defined in Section 3.4.10.3 on page 29.	20
De	escription	
Th ca <i>sa</i>	nis function frees the memory to which <i>notification</i> points. This memory was allo- ited by the Message Service library in a previous call to the MsgQueueGroupTrack() function.	25
Fo sp	or details, refer to the description of the <i>notificationBuffer</i> parameter in the corre- conding invocation of the <i>saMsgQueueGroupTrack()</i> function.	
Re	eturn Values	
SA	A_AIS_OK - The function completed successfully.	30
SA co	A_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as prruption). The library cannot be used anymore.	
SA ruj	A_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- pted, uninitialized, or has already been finalized.	35
SA	A_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	
SA thi	A_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on is cluster node because it is not a member node.	40



	See Also saMsgQueueGroupTrack()	1
3.8 N	Message Send and Receive Operations	5
3.8.1	saMsgMessageSend() and saMsgMessageSendAsync()	
	Prototype	
	SaAisErrorT saMsgMessageSend(SaMsgHandleT msgHandle,	10
	const SaNameT *destination, const SaMsgMessageT *message, SaTimeT timeout	15
);	
	SaAisErrorT saMsgMessageSendAsync(SaMsgHandleT msgHandle, SaInvocationT invocation,	20
	const SaNameT *destination, const SaMsgMessageT *message, SaMsgAckFlagsT ackFlags	25
);	20
	Parameters	30
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	35
	<i>invocation</i> - [<i>in</i>] This parameter associates this invocation of <i>saMsgMessageSendAsync(</i>) with a corresponding invocation of the <i>saMsgMessageDeliveredCallback(</i>) function. This parameter is ignored if <i>ackFlags</i> is set to zero. The <i>SaInvocationT</i> type is defined in [1].	40

5

15

20

25

30

35

40

destination - [in] A pointer to the name of a message queue or message queue group	
to which the message referred to by <i>message</i> 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 10 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 gueue or message gueue 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 <i>saMsgMessageDeliveredCallback()</i> is not invoked, and the caller is not informed if an error occurs. If the value of the <i>ackFlags</i> field is set to SA_MSG_MESSAGE_DELIVERED_ACK, and this call returns SA_AIS_OK, <i>saMsgMessageDeliveredCallback()</i> 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 <i>saMsgMessageDeliveredCallback()</i> when it invoked the <i>saMsgInitialize()</i> 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	10
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.	15
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 mes- sage queue; if the return value is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been delivered to any destina- tion message queue. 	25
 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. 	30
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	35

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 40 saMsgMessageDeliveredCallback() function.


Return Values	
SA_AIS_OK - The fu	inction completed successfully.
SA_AIS_ERR_LIBRA corruption). The libra	ARY - An unexpected problem occurred in the library (such as iry cannot be used anymore.
SA_AIS_ERR_TIME timeout defined by the is unspecified whether applies only to <i>saMs</i>	OUT - An implementation-dependent timeout occurred, or the ne <i>timeout</i> parameter occurred before the call could complete. It er the call succeeded or whether it did not. This error code <i>gMessageSend()</i> .
SA_AIS_ERR_TRY_ cess may retry later.	AGAIN - The service cannot be provided at this time. The pro-
SA_AIS_ERR_BAD_ rupted, uninitialized,	_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- or has already been finalized.
SA_AIS_ERR_INIT - sage Service was inc function is missing, a in <i>ackFlags</i> of <i>saMsg</i>	The previous invocation of <i>saMsgInitialize()</i> to initialize the Mescomplete, since the <i>saMsgMessageDeliveredCallback()</i> callback and the user specified SA_MSG_MESSAGE_DELIVERED_ACK <i>gMessageSendAsync()</i> .
SA_AIS_ERR_INVA	LID_PARAM - A parameter is not set correctly.
SA_AIS_ERR_NO_M the service is out of r	MEMORY - Either the Message Service library or the provider of memory and cannot provide the service.
SA_AIS_ERR_NO_F memory).	RESOURCES - There are insufficient resources (other than
SA_AIS_ERR_TOO_ to be sent exceeds the Refer to the enum SA on page 34.	_BIG - This value is returned if the message size of the message he maximum message size supported by the implementation. A_MSG_MAX_MESSAGE_SIZE_ID, defined in Section 3.4.13
SA_AIS_ERR_NOT_ queue group name d found.	_EXIST - The destination message queue name or message lesignated by the name to which <i>destination</i> points cannot be
SA_AIS_ERR_QUE message queue, the queue group, the sel queue group, all sele	JE_FULL - If the name to which <i>destination</i> points refers to a message queue is full. If the name refers to a unicast message ected member queue is full. If it refers to a multicast message ected member message queues are full.
SA_AIS_ERR_QUE parameter points des is empty.	JE_NOT_AVAILABLE - The name to which the <i>destination</i> signates a message queue group, and the message queue group



SA_AIS_ERR_BAD_FLAGS - The ackFlags parameter is invalid.	1
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	5
saMsgMessageSendReceive(), saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgMessageGet(), SaMsgMessageDeliveredCallbackT	10
SaMsgMessageDeliveredCallbackT	10
Prototype	
typedef void (*SaMsgMessageDeliveredCallbackT)(15
SaInvocationT invocation,	
SaAisErrorT error	
);	20
Parameters	20
<i>invocation</i> - [<i>in</i>] A designator that associates this invocation to a previous call to one of the <i>saMsgMessageSendAsync()</i> or <i>saMsgMessageReplyAsync()</i> functions. The <i>SaInvocationT</i> type is defined in [1].	25
<i>error</i> - [<i>in</i>] This parameter specifies whether the message sent by the corresponding invocation of <i>saMsgMessageSendAsync()</i> or <i>saMsgMessageReplyAsync()</i> has been delivered to the destination message queue(s) or to the reply buffer supplied by the process that invoked the <i>saMsgMessageSendReceive()</i> function. The <i>SaAisErrorT</i> type is defined in [1].	30
 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. 	35
 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. 	40
SA AIS ERR RAD HANDLE. The handle meriliandle specified in the corre	
	 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(), 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.

	<i>saMsgMessageReplyAsync()</i> functions is invalid, since it is corrupted, unini- tialized, or has already been finalized.	1
•	SA_AIS_ERR_INVALID_PARAM - A parameter was not set correctly in the corresponding invocation of the <i>saMsgMessageSendAsync()</i> or <i>saMsgMessageReplyAsync()</i> functions.	5
•	SA_AIS_ERR_NO_MEMORY - Either the Message Service library or the pro- vider of the service is out of memory and cannot provide the service.	
•	SA_AIS_ERR_NO_RESOURCES - Insufficient resources (other than mem- ory).	10
•	SA_AIS_ERR_TOO_BIG - This value is returned if	
	 the message size of the message to be sent in the corresponding invocation of <i>saMsgMessageSendAsync()</i> exceeds the maximum message size sup- ported by the implementation (refer to the enum SA_MSG_MAX_MESSAGE_SIZE_ID, defined in Section 3.4.13 on page 34), or 	15
	 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. 	20
	SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group designated by the name to which <i>destination</i> points in the corresponding invocation of <i>saMsgMessageSendAsync()</i> cannot be found, or the reply buffer identified by <i>*senderld</i> in the corresponding invocation of <i>saMsgMessageReplyAsync()</i> cannot be located.	25
•	SA_AIS_ERR_NO_SPACE - The reply buffer to which <i>senderld</i> points in the corresponding invocation of <i>saMsgMessageReplyAsync()</i> is not large enough to contain the reply message.	0.0
	SA_AIS_ERR_QUEUE_FULL - If the name to which <i>destination</i> points in the corresponding invocation of the <i>saMsgMessageSendAsync()</i> 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	30
	name refers to a multicast message queue group, all selected member mes- sage queues are full.	35
•	SA_AIS_ERR_QUEUE_NOT_AVAILABLE - The <i>destination</i> parameter in the corresponding invocation of the <i>saMsgMessageSendAsync()</i> function points	
	to the name of a message queue group, and the message queue group is empty.	40



 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. 	1 5
Description	
The Message Service invokes this callback to indicate whether a previous call to saMsgMessageSendAsync() or saMsgMessageReplyAsync() could deliver a mes- sage to the destination successfully.	10
This callback is invoked in the context of a thread calling <i>saMsgDispatch()</i> on the handle <i>msgHandle</i> that was specified in the corresponding <i>saMsgMessageSendAsync()</i> or <i>saMsgMessageReplyAsync()</i> call.	15
If an error occurs, it is returned in the error parameter.	15
During this call or after this call returns, the process may deallocate the memory for	
 either the data buffer to which <i>message->data</i> points, and which was passed pre- viously to the corresponding <i>saMsgMessageSendAsync()</i> function or for the data buffer to which <i>replyMessage->data</i> points, and which was passed previously to the corresponding <i>saMsgMessageReplyAsync()</i> call. 	20
Message delivery properties:	
For <i>saMsgMessageSendAsync()</i> , 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.	25
 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. 	30
 Multicast message queue group: If <i>error</i> is SA_AIS_OK, the message has been successfully delivered to at least one member of the message queue group; if <i>error</i> 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. 	35
For <i>saMsgMessageReplyAsync()</i> , the message delivery properties apply to the delivery of the reply message into the reply buffer supplied by the process that invoked the <i>saMsgMessageSendReceive()</i> function. If <i>error</i> is SA_AIS_OK, the message has	40



been successfully delivered; if <i>error</i> is neither SA_AIS_ERR_LIBRARY nor SA_AIS_ERR_TIMEOUT, the message has not been and will not be delivered.	1
Return Values	
None	5
See Also	
saMsgMessageSendAsync(), saMsgMessageReplyAsync(), saMsgMessageSendReceive(), saMsgDispatch()	10
3.8.3 saMsgMessageGet()	
Prototype	
SaAisErrorT saMsgMessageGet(15
SaMsgQueueHandleT queueHandle,	
SaMsgMessageT *message,	
SaTimeT *sendTime,	20
SaMsgSenderIdT *senderId,	
SaTimeT timeout	
);	25
Parameters	
<i>queueHandle</i> - [<i>in</i>] The handle of the message queue from which a message is to be received. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	30
<i>message</i> - [<i>in/out</i>] A pointer to a message structure of type <i>SaMsgMessageT</i> (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 	35
parameter. If <i>data</i> as an <i>in</i> parameter is NULL, the value of <i>size</i> 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 <i>saMsgMessageDataFree()</i> function.	40

 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. 	1
 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. 	10
If <i>senderName</i> as an <i>in</i> parameter is NULL, no sender name is provided to the caller.	15
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 <i>saMsgMessageSend()</i> , <i>saMsgMessageSendAsync()</i> , or <i>saMsgMessageSendReceive()</i> . If <i>sendTime</i> as an <i>in</i> parameter is NULL, it is ignored.	20
senderId - [<i>in</i> /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	25
"senderia.	30
<i>timeout</i> - [<i>in</i>] The time duration that specifies how long the <i>saMsgMessageGet()</i> function waits for the arrival of a message before it returns with a timeout error. If <i>timeout</i> is set to 0, the call returns immediately; if a message has been found in the message gueue before the call returns, and no other error has occurred, this mes-	
sage is returned, and the return value is set to SA_AIS_OK; otherwise, SA_AIS_ERR_TIMEOUT is returned. The <i>SaTimeT</i> type is defined in [1].	35
Description	
This function retrieves a message from the message queue designated by	40
guave Handle. This function will block until a manager is available to be retrieved, or	

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 <i>saMsgMessageGet()</i> 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.	1
When the Message Service library allocates the buffer to which <i>data</i> points, the invoking process must deallocate this buffer by calling the <i>saMsgMessageDataFree()</i> function.	5
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.	10
Return Values	15
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 <i>timeout</i> parameter occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	20
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	25
SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> 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. 	30
 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 par- ticular, this value is returned if <i>senderId</i> is NULL.	35
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).	40



	SA_AIS_ERR_NO_SPACE - The message could not be received because the buffer provided was not large enough.	1
	SA_AIS_ERR_INTERRUPT - This error code is returned if the call is terminated by a call to <i>saMsgMessageCancel()</i> .	5
	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()	10
3.8.4 s	saMsgMessageDataFree()	15
	Prototype	10
	SaAisErrorT saMsgMessageDataFree(
	SaMsgHandleT msgHandle,	20
	void *data	
);	
	Parameters	25
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	
	<i>data</i> - [<i>in</i>] A pointer to the buffer that was allocated by <i>saMsgMessageGet()</i> or <i>saMsgMessageSendReceive()</i> and is to be deallocated.	30
	Description	
	This function frees the memory to which <i>data</i> points. This memory was allocated by the Message Service library in a previous call to the <i>saMsgMessageGet()</i> or <i>saMsgMessageSendReceive()</i> functions.	35
	For details, refer to the description of the <i>message</i> parameter in the corresponding invocation of the <i>saMsgMessageGet()</i> function and to the description of the <i>receiveMessage</i> parameter in the corresponding invocation of the <i>saMsgMessageSendReceive()</i> function.	40



	Return Values	1
	SA_AIS_OK - The function completed successfully.	
	SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	5
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	
	SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.	4.0
	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	10
	See Also	
	saMsgMessageGet(),	15
3.8.5	SaMsgMessageReceivedCallbackT	
	Prototype	
		20
	typedef void ("SamsgmessageReceivedCalibackT)(
	SaMsgQueueHandleT queueHandle	
);	25
	Parameters	25
	<i>queueHandle</i> - [<i>in</i>] The handle to the message queue from which the message can be received. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	30
	Description	
	The Message Service invokes this callback function to notify a process that a mes- sage can be received from the message queue designated by <i>queueHandle</i> .	05
	This callback is invoked in the context of a thread calling <i>saMsgDispatch()</i> on the handle <i>msgHandle</i> that was specified when the process invoked one of the <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> functions to obtain the handle <i>queueHandle</i> .	35
	The process can receive this message by invoking the <i>saMsgMessageGet()</i> function.	40
	The Message Service invokes this callback whenever a message is placed in the message queue, irrespective of its priority.	



	Return Values	1
	None	
	See Also	5
	saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync(), saMsgMessageSend(), saMsgMessageSendAsync(), saMsgMessageSendReceive(), saMsgMessageReply(), saMsgMessageReplyAsync(), saMsgDispatch()	10
3.8.6 s	saMsgMessageCancel()	10
	Prototype	
	SaAisErrorT saMsgMessageCancel(15
	SaMsgQueueHandleT queueHandle	
);	
	Parameters	20
	<i>queueHandle</i> - [<i>in</i>] The handle to the message queue for which blocking calls to <i>saMsgMessageGet()</i> are to be canceled. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	
	Description	25
	This function cancels all blocking calls to <i>saMsgMessageGet()</i> in the invoking process for the message queue designated by <i>queueHandle</i> .	
	This function is normally called during fault recovery.	30
	The canceled call returns with the error code set to SA_AIS_ERR_INTERRUPT.	
	If no process is blocking in an <i>saMsgMessageGet()</i> call, the <i>saMsgMessageCancel()</i> call has no effect and returns SA_AIS_ERR_NOT_EXIST.	
	Return Values	35
	SA_AIS_OK - The function completed successfully.	
	SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	40



 SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later. SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> is invalid, due to one or both of the reasons below: It is corrupted, was not obtained with the <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenCallback()</i> functions, or the corresponding message queue has already been closed. The handle <i>msgHandle</i> that was passed to the functions <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> has already been finalized. SA_AIS_ERR_NOT_EXIST - No process was blocking in an <i>saMsgMessageGet()</i> 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 20	SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	1
 SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> is invalid, due to one or both of the reasons below: It is corrupted, was not obtained with the <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenCallback()</i> functions, or the corresponding message queue has already been closed. The handle <i>msgHandle</i> that was passed to the functions <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> has already been finalized. SA_AIS_ERR_NOT_EXIST - No process was blocking in an <i>saMsgMessageGet()</i> 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 20 saMsgMessageGet(), <i>saMsgQueueOpen()</i>, <i>saMsgQueueOpenAsync()</i>	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	5
 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() 	SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> is invalid, due to one or both of the reasons below:	
 The handle <i>msgHandle</i> that was passed to the functions <i>saMsgQueueOpen()</i> or <i>saMsgQueueOpenAsync()</i> has already been finalized. SA_AIS_ERR_NOT_EXIST - No process was blocking in an <i>saMsgMessageGet()</i> 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() 	 It is corrupted, was not obtained with the saMsgQueueOpen() or saMsgQueueOpenCallback() functions, or the corresponding message queue has already been closed. 	10
SA_AIS_ERR_NOT_EXIST - No process was blocking in an saMsgMessageGet() call. 15 SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node. 20 See Also 20 saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync() 25	 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. 20 See Also saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync() 25	SA_AIS_ERR_NOT_EXIST - No process was blocking in an <i>saMsgMessageGet()</i> call.	15
20 See Also saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync() 25	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	
saMsgMessageGet(), saMsgQueueOpen(), saMsgQueueOpenAsync() 25	See Also	20
25	saMsgMessageGet(),	
		25



3.9 Request-Reply Operations	1
3.9.1 saMsgMessageSendReceive()	
Prototype	5
SaAisErrorT saMsgMessageSendReceive(SaMsgHandleT msgHandle,	
const SaNameT *destination, const SaMsgMessageT *sendMessage, SaMsgMessageT *receiveMessage,	10
SaTimeT *replySendTime, SaTimeT timeout);	15
Parameters <i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	20
<i>destination</i> - [<i>in</i>] A pointer to the name of a message queue or a unicast message queue group to which the message pointed to by <i>sendMessage</i> is to be sent. It is not permitted to specify a multicast message queue group in <i>destination</i> . The <i>SaNameT</i> type is defined in [1].	25
<i>sendMessage</i> - [<i>in</i>] A pointer to the message structure of <i>SaMsgMessageT</i> type (as defined in Section 3.4.11 on page 30) for the message to be sent. The message structure consists of a <i>type</i> field that contains the message type, a <i>version</i> field that is used to distinguish different versions of a message of the same type, a <i>data</i> field that points to the message data, a size field that contains the size of the message data.	30
<i>priority</i> field that gives the priority of the message, and a <i>senderName</i> pointer. If <i>senderName</i> is not NULL, it points to an area which contains the sender name, and which is supplied by the caller; If <i>senderName</i> is NULL, the sending process does not provide its sender name, and a receiving process expecting a sender name will get <i>message->senderName->length</i> set to zero.	35



<i>receiveMessage</i> - [<i>in/out</i>] A pointer to the message structure of type <i>SaMsgMessageT</i> , as defined in Section 3.4.11 on page 30. This message structure contains the following fields:	1
 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. 	5
If <i>data</i> as an <i>in</i> parameter is NULL, the value of <i>size</i> 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 <i>saMsgMessageDataFree()</i> function.	10
 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. 	15
 type - [out] The message type of the received message. 	
 version - [out] Distinguishes different versions of a message of the same type. 	20
 priority - [out] The priority of the received message. 	20
 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. 	25
<i>replySendTime</i> - [<i>in/out</i>] If <i>replySendTime</i> as an <i>in</i> parameter is not NULL, it points to a value of type <i>SaTimeT</i> (as defined in [1]) on successful return from <i>saMsgMessageSendReceive()</i> . This value represents the timestamp when the reply message was written into the buffer to which <i>receiveMessage->data</i> points. If <i>replySendTime</i> as an <i>in</i> parameter is NULL, it is ignored.	30
<i>timeout</i> - [<i>in</i>] The time duration that specifies how long the <i>saMsgMessageSendReceive()</i> function must wait before it returns with a timeout error. The <i>SaTimeT</i> type is defined in [1].	35
Description	40

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 <i>sendMessage</i> points is sent to the message queue or message queue group designated by the name to which <i>destination</i> points. The message is sent synchronously, that is, the process blocks waiting for a reply. The	1
reply consists of the message referred to by <i>receiveMessage</i> . It must be sent by invoking one of the <i>saMsgMessageReply()</i> or <i>saMsgMessageReplyAsync()</i> functions, and it must arrive before the time duration specified by the <i>timeout</i> parameter elapses; If the reply does not arrive during this time, the error SA_AIS_ERR_TIMEOUT is returned.	5
In absence of errors, the thread that invoked <i>saMsgMessageSendReceive()</i> will receive the corresponding reply, even when other threads are waiting for replies to other invocations of <i>saMsgMessageSendReceive()</i> . Moreover, the reply message sent by invoking <i>saMsgMessageReply()</i> or <i>saMsgMessageReplyAsync()</i> is not enqueued in a message queue.	10
After this call returns, the invoking process may deallocate the memory for the buffer to which <i>sendMessage->data</i> points. When the buffer to which <i>receiveMessage->data</i> points is allocated by the Message Service library, the invoking process must deallocate this buffer space promptly by calling the <i>saMsgMessageDataFree()</i> function.	15
Message delivery properties:	20
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.	25
If SA_AIS_ERR_NO_SPACE is returned, the message has been delivered to a pro- cess receiving from the destination message queue, because this error applies only when receiving the reply.	30
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.	35
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	40

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.	1
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred, or the timeout defined by the <i>timeout</i> parameter occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	5
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	10
SA_AIS_ERR_INVALID_PARAM - One of the parameters is not set correctly. In par- ticular, this applies if the <i>destination</i> parameter points to the name of a multicast mes- sage 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.	15
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.	20
SA_AIS_ERR_NOT_EXIST - The destination message queue or message queue group designated by the name to which <i>destination</i> points cannot be found.	25
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 <i>destination</i> points to the name of a message queue, the message queue is full. If <i>destination</i> points to the name of a unicast message queue group, the selected member message queue is full.	30
SA_AIS_ERR_QUEUE_NOT_AVAILABLE - The <i>destination</i> parameter points to a message queue group, and the message queue group is empty.	35
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()	40



3.9.2	9.2 saMsgMessageReply() and saMsgMessageReplyAsync()			
	Prototype			
	SaAisErrorT saMsgMessageReply(5		
	SaMsgHandleT msgHandle,	0		
	const SaMsgMessageT *replyMessage,			
	const SaMsgSenderIdT *senderId,			
	SaTimeT timeout	10		
);			
	SaAisErrorT saMsgMessageReplyAsync(
	SaMsgHandleT msgHandle,	15		
	SaInvocationT invocation,	10		
	const SaMsgMessageT *replyMessage,			
	const SaMsgSenderIdT *senderId,			
	SaMsgAckFlagsT ackFlags	20		

);

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 Message3540

If senderName is not NULL, it points to an area which contains the sender name, and



which is supplied by the caller; if <i>senderName</i> 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 <i>saMsgMessageSendReceive()</i> call) requested a sender name, the <i>saMsgMessageSendReceive()</i> call sets the sender name to zero (<i>receiveMessage->senderName->length</i> = 0).	1 5
<i>senderId</i> - [<i>in</i>] Pointer to the same value that the <i>saMsgMessageGet()</i> call obtained in * <i>senderId</i> for the message to which the caller is replying. The <i>SaMsgSenderIdT</i> type is defined in Section 3.4.2 on page 23.	10
ackFlags - [<i>in</i>] 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	10
SA_MSG_MESSAGE_DELIVERED_ACK, the caller requires to be acknowledged whether the message has been stored in the reply buffer provided in the correspond- ing <i>saMsgMessageSendReceive()</i> 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 SaMagAg/ElagoT type is defined in Section 2.4.4 on page 24	15
<i>timeout</i> - [<i>in</i>] The <i>saMsgMessageReply()</i> invocation is considered to have failed if it does not complete within the specified duration. The <i>SaTimeT</i> type is defined in [1].	20
Description	
These functions are used to reply to a message that was previously sent by calling the <i>saMsgMessageSendReceive()</i> function in which case the Message Service sets <i>*senderld</i> for the message received by the <i>saMsgMessageGet()</i> function to a non-zero value. This indicates to the process that it must reply to the received message by invoking either <i>saMsgMessageReply()</i> or <i>saMsgMessageReplyAsync()</i> and with <i>*senderld</i> set to the same value that the <i>saMsgMessageGet()</i> function set for <i>*senderld</i> .	25 30
A process may not reply to a message more than once.	
The <i>saMsgMessageReply()</i> function waits synchronously (that is, it blocks) until the reply message has been placed in the reply buffer provided by the process that invoked <i>saMsgMessageSendReceive()</i> , or an error occurs. After the <i>saMsgMessageReply()</i> function returns, the invoking process may deallocate the memory for the buffer to which <i>replyMessage->data</i> points.	35
The function <i>saMsgMessageReplyAsync()</i> returns as soon as possible, without wait- ing until the reply message has been placed in the reply buffer provided by the pro- cess that invoked <i>saMsgMessageSendReceive()</i> . If the value of the <i>ackFlags</i> field is	40



zero, the <i>saMsgMessageDeliveredCallback()</i> is not invoked, and the caller is not informed whether an error occurred or whether it did not.	1
If the value of the <i>ackFlags</i> field is set to SA_MSG_MESSAGE_DELIVERED_ACK, and this call returns SA_AIS_OK, <i>saMsgMessageDeliveredCallback()</i> is invoked to indicate whether the message was delivered or whether an error occurred. For this purpose, the invoking process must have supplied the <i>saMsgMessageDeliveredCallback()</i> when it called the <i>saMsgInitialize()</i> function. If <i>saMsgMessageReplyAsync()</i> returns successfully, and <i>ackFlags</i> is not set to zero,	5
the caller may deallocate the memory for the buffer to which replyMessage->data points either during the invocation of saMsgMessageDeliveredCallback() or after saMsgMessageDeliveredCallback() returns.	10
If <i>saMsgMessageReplyAsync()</i> returns an error, or <i>ackFlags</i> is set to zero (meaning that <i>saMsgMessageDeliveredCallback()</i> will not be called), the caller may deallocate the memory for the buffer to which <i>replyMessage->data</i> points as soon as <i>saMsgMessageReplyAsync()</i> returns.	15
If the process that invoked <i>saMsgMessageSendReceive()</i> exits or calls <i>saMsgFinalize()</i> for its handle <i>msgHandle</i> before <i>saMsgMessageReply()</i> or <i>saMsgMessageReplyAsync()</i> completes, the Message Service returns SA_AIS_ERR_NOT_EXIST to the process that invoked <i>saMsgMessageReply()</i> or <i>saMsgMessageReplyAsync()</i> .	20
Message delivery properties:	
<i>saMsgMessageReply</i> : If the return value is SA_AIS_OK, the Message Service delivers the reply to the pro- cess that invoked <i>saMsgMessageSendReceive()</i> . 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	25
reply.	30
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	35
saMsgMessageDeliveredCallback() function for the message delivery properties.	
Return Values	40

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.	1
SA_AIS_ERR_TIMEOUT - Before the call could complete, either an implementation- dependent timeout occurred, or the timeout specified by the <i>timeout</i> parameter in the <i>saMsgMessageReply()</i> call occurred. It is unspecified whether the <i>saMsgMessageReply()</i> call succeeded or whether it did not.	5
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	
SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	10
SA_AIS_ERR_INIT - The previous invocation of <i>saMsgInitialize()</i> to initialize the Mes- sage Service was incomplete, since the <i>saMsgMessageDeliveredCallback()</i> callback function is missing, and the user specified SA_MSG_MESSAGE_DELIVERED_ACK in <i>ackFlags</i> of <i>saMsgMessageReplyAsync()</i> .	15
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 <i>saMsgMessageSendReceive()</i> .	20
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).	25
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.	20
SA_AIS_ERR_NOT_EXIST - Either no thread is waiting for a reply, or the value to which the <i>senderId</i> parameter points is invalid.	30
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.	35
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	40
saMsgMessageSendReceive(),	



3.10 Set and Get Critical Capacity Thresholds Operations	1
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()	5
Prototype	
SaAisErrorT saMsgQueueCapacityThresholdsSet(SaMsgQueueHandleT queueHandle, const SaMsgQueueThresholdsT *thresholds	10
); Parameters	15
<i>queueHandle</i> - [<i>in</i>] The handle to the message queue for which the critical capacity thresholds are to be set. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	20
<i>thresholds</i> - [<i>in</i>] Pointer to an <i>SaMsgQueueThresholdsT</i> structure (as defined in Sec- tion 3.4.12.2 on page 32) that contains the critical capacity thresholds to be set for the message queue identified by the handle <i>queueHandle</i> .	
Description	25
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>i</i> must fulfill the following relationships:	30
0 <= capacityAvailable[i] <= capacityReached[i] <= size[i],	
where <i>size</i> represents the array containing the sizes of the priority areas as specified at creation time, and <i>capacityReached</i> and <i>capacityAvailable</i> are arrays in the struc- ture to which <i>thresholds</i> points and which contain for each priority area the high-water capacity threshold and the low-water capacity threshold respectively.	35
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.	40



SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	1
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	5
SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> 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. 	10
 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:	15
The <i>thresholds</i> pointer is NULL.	
 The values set for each priority area <i>i</i> do not fulfill the following relationships: 0 <= capacityAvailable[<i>i</i>] <= capacityReached[<i>i</i>] <= size[<i>i</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 prior- ity areas as specified at creation time. 	20
SA_AIS_ERR_VERSION - The invoked function is not supported in the version spec- ified in the call to initialize this instance of the Message Service library.	25
SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	
See Also	30
saMsgQueueCapacityThresholdsGet(), saMsgQueueOpen(), saMsgQueueOpenAsync()	
	35

SERVICE AVAILABILITY FORUM

3.10.2 saMsgQueueCapacityThresholdsGet()	1
Prototype	
SaAisErrorT saMsgQueueCapacityThresholdsGet(5
SaMsgQueueHandleT queueHandle,	
SaMsgQueueThresholdsT *thresholds	
);	10
Parameters	10
<i>queueHandle</i> - [<i>in</i>] The handle to the message queue for which the critical capacity thresholds are to be retrieved. The <i>SaMsgQueueHandleT</i> type is defined in Section 3.4.1.2 on page 23.	15
<i>thresholds</i> - [<i>out</i>] Pointer to an <i>SaMsgQueueThresholdsT</i> 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 <i>queueHandle</i> .	20
Description	
This function enables a user application to retrieve the critical capacity thresholds for a message queue.	
Return Values	25
SA_AIS_OK - The function completed successfully.	
SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.	30
SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.	
SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	35
SA_AIS_ERR_BAD_HANDLE - The handle <i>queueHandle</i> 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. 	40



	 The handle msgHandle that was passed to the functions saMsgQueueOpen() or saMsgQueueOpenAsync() has already been finalized. 	1
:	SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. This error is returned if the <i>thresholds</i> pointer is NULL.	5
i	SA_AIS_ERR_VERSION - The invoked function is not supported in the version spec- ified in the call to initialize this instance of the Message Service library.	Ū
: 1	SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node because it is not a member node.	10
:	See Also	
:	saMsgQueueCapacityThresholdsSet(), saMsgQueueOpen(), saMsgQueueOpenAsync()	15
3.11 R	Retrieve Metadata Size and Limit Fetch Operations	
- 1	The <i>saMsgMetadataSizeGet()</i> and the <i>saMsgLimitGet()</i> functions retrieve implemen- tation-specific values.	
3.11.1	saMsgMetadataSizeGet()	20
ļ	Prototype	
	SaAisErrorT saMsgMetadataSizeGet(25
	SaMsgHandleT msgHandle,	
	SaUint32T *metadataSize	
,);	30
I	Parameters	30
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	35
	<i>metadataSize</i> - [<i>out</i>] 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].	
	5	40



5

10

25

30

35

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;	15
SaUint32T	version;	
SaUint16T	senderNameLength;	
SaUint8T	priority;	20
SaSizeT	dataSize;	20

)

}

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 40 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.	1
	SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The pro- cess may retry later.	5
	SA_AIS_ERR_BAD_HANDLE - The handle <i>msgHandle</i> is invalid, since it is cor- rupted, uninitialized, or has already been finalized.	
	SA_AIS_ERR_INVALID_PARAM - The metadataSize pointer is NULL.	10
	SA_AIS_ERR_VERSION - The invoked function is not supported in the version spec- ified 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.	15
	See Also	-
	saMsgInitialize(), saMsgQueueOpen(), saMsgQueueStatusGet()	
3.11.2	saMsgLimitGet()	20
	Prototype	
	SaAisErrorT saMsgLimitGet(
	SaMsgHandleT msgHandle,	25
	SaMsgLimitIdT limitId,	
	SaLimitValueT *limitValue	
);	30
	Parameters	
	<i>msgHandle</i> - [<i>in</i>] The handle which was obtained by a previous invocation of the <i>saMsgInitialize()</i> function and which designates this particular initialization of the Message Service. The <i>SaMsgHandleT</i> type is defined in Section 3.4.1.1 on page 23.	35
	<i>limitId</i> - [<i>in</i>] The Message Service limit whose implementation-specific value needs to be obtained. The limits are defined in the <i>SaMsgLimitIdT</i> type in Section 3.4.13 on page 34.	40
	<i>limitValue</i> - [<i>out</i>] Pointer to the current value of the limit specified in <i>limitId</i> . For details regarding this type, refer to the SA Forum Overview document ([1]).	40

5

10

15

20

25

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. 30

See Also

saMsgInitialize()

35



4	Message Service Information Model	1
	The Message Service information model is described in UML and has been orga- nized in a UML class diagram.	F
	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]).	5
	The Message Service UML class diagram has three classes, which show the con- tained attributes.	10

4.1 DN Format for the Message Service UML Classes

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

Table 2 DN Formats for Objects of Message Service Classes

The '*' notation at the end of a DN format indicates that zero, one or more RDNs may 25 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]).

15

30



FIGURE 1	Message Service UML Classes	

< <runtime>> SaMsgQueue</runtime>	5
safMq : SaStringT [1]{RDN, RUNTIME, CACHED} saMsgQueueIsPersistent : SaBooIT [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 : SaBooIT [1]{RUNTIME, CACHED, SAUINT32T}	10
< <runtime>></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} saMsgQueuePriorityQapacityAvailable : SaUint64T [1]{RUNTIME, CACHED}	20
saMsgQueuePriorityCapacityReached : SaUint64T [1]{RUNTIME, CACHED}	25
< <runtime>> SaMsgQueueGroup</runtime>	
safMqg : SaStringT [1]{RDN, RUNTIME, CACHED} saMsgQueueGroupPolicy : SaMsgQueueGroupPolicyT [1]{RUNTIME, CACHED, SAUINT32T} saMsgQueueGroupNumQueues : SaUint32T [1]{RUNTIME, CACHED} saMsgQueueGroupMemberName : SaNameT [0*]{RUNTIME, CACHED}	30

35



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



5
10
15
20
25
30
35



6	Alarms and Notifications	1
	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 	5
	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 necessar- ily require an operator intervention.	10
	The recommended vehicle to be used for producing alarms and notifications is the Notification Service of the Service Availability [™] Forum (abbreviated to NTF, see [2]), and hence the various notifications are partitioned into categories as described in this service.	15
	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 perspec- tive (see [7]).	20 25
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.	30
	 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 <i>saNtfNotificationSend()</i> API during the production of the actual alarm. 	35
	 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 	40

Notification Service.



 NCI id - The notification class identifier is an attribute of type SaNtfClassIdT. The vendorld portion of the SaNtfClassIdT data structure must be set to SA_NTF_VENDOR_ID_SAF always. The majorld and minorld will vary based on the specific SA Forum service and the particular notification. Every SA Forum service shall have a majorld as described in the enumeration SaNtfSafServicesT of the Notification Service specification. 	1
 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 <i>safApp</i> RDN value portion of the DN set to the specified standard RDN value of the SA Forum AIS naming the notification. For details on the SA Forum AIS naming convention, refer to the SA Forum Overview document ([1]). 	10
	15
6.2 Message Service Notifications	
The following sections describe a set of notifications that a Message Service imple- mentation shall produce.	
The notifying object must be set to the DN " <i>safApp=safMsgService</i> " for the Message Service.	20
The value of the <i>majorld</i> field in the notification class identifier (<i>SaNtfClassIdT</i>) must be set to SA_SVC_MSG (as defined in the <i>SaServicesT</i> enum in [1]) in all notifica-tions generated by the Message Service.	25
The <i>minorld</i> field within the notification class identifier (<i>SaNtfClassIdT</i>) is set distinctly for each individual notification as described below. This field is range-bound, and the used ranges are:	30
• Alarms: (0x01–0x64)	00
 State change notifications: (0x65–0xC8) 	
 Object change notifications: (0xC9–0x12C) 	
 Attribute change notifications: (0x12D–0x190) 	35
6.2.1 Message Service Alarms	
The Message Service does not issue any alarms at the time of publication of this specification.	40



6.2.2 Message Serv	vice State Change Notifi	cations	1
6.2.2.1 Message Que	eue Capacity Reached		
Description			5
All priority are Section 3.4.1	eas of the message queue 2.1 on page 32).	e are filled up to their critical capacities (see	
Table	e 3 Message Queue Cap	acity Reached Notification	10
NTF Attribute Name	Attribute Type (NTF- Recommended Value)	SA Forum-Recommended Value	
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE	15
Notification Object	Mandatory	LDAP DN of the message queue (as speci- fied in Section 4.1) that is full and cannot accept any more messages.	
Notification Class Identifier	NTF-internal	minorld = 0x65	20
Additional Text	Optional	Unspecified	
Additional Informa- tion	Optional	Unspecified	25
Source Indicator	Mandatory	SA_NTF_OBJECT_OPERATION or SA_NTF_UNKNOWN_OPERATION	
Changed State Attribute ID	Optional	SA_MSG_DEST_CAPACITY_STATUS	30
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).				
			10	
NTF Attribute Name	Recommended Value)	SA Forum-Recommended Value		
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE	15	
Notification Object	Mandatory	LDAP DN of the message queue (as speci- fied in Section 4.1) which is available for receipt of messages.	10	
Notification Class Identifier	NTF-internal	<i>minorld</i> = 0x66	20	
Additional Text	Optional	Unspecified		
Additional Informa- tion	Optional	Unspecified	25	
Source Indicator	Mandatory	SA_NTF_OBJECT_OPERATION or SA_NTF_UNKNOWN_OPERATION		
Changed State Attribute ID	Optional	SA_MSG_DEST_CAPACITY_STATUS	30	
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						
Descri	ption					
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).						
Та	ble 5 Message Queue G	Froup Capacity Reached Notification				
NTF Attribute Name	Attribute Type (NTF- Recommended Value)	SA Forum-Recommended Value	10			
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE				
Notification Object	Mandatory	LDAP DN of the message queue group (as speci- fied in Section 4.1) that is full and cannot accept any more messages.	15			
Notification Class Identi- fier	NTF-internal	minorld = 0x67	20			
Additional Text	Optional	Unspecified				
Additional Information	Optional	Unspecified	25			
Source Indica- tor	Mandatory	SA_NTF_OBJECT_OPERATION or SA_NTF_UNKNOWN_OPERATION				
Changed State Attribute ID	Optional	SA_MSG_DEST_CAPACITY_STATUS	30			
Old Attribute Value	Optional	Unspecified				
New Attribute Value	Mandatory	SA_MSG_QUEUE_GROUP_CAPACITY_REACHED	35			

6.2.2.4 Message Queue Group Capacity Available			1
Descrip	otion		
At least longer f Reache tion 3.4 Tab	one priority area in one n illed up to its critical capa d condition has been noti .12.1 on page 32).	nessage queue in the message queue group is no city after the Message Queue Group Capacity fied for the message queue group before (see Sec-	5 10
NTF Attribute Name	Attribute Type (NTF- Recommended Value)	SA Forum-Recommended Value	
Event Type	Mandatory	SA_NTF_OBJECT_STATE_CHANGE	15
Notification Object	Mandatory	LDAP DN of the message queue group (as speci- fied in Section 4.1) that is available to receive messages.	
Notification Class Identifier	NTF-internal	minorld = 0x68	20
Additional Text	Optional	Unspecified	
Additional Infor- mation	Optional	Unspecified	25
Source Indica- tor	Mandatory	SA_NTF_OBJECT_OPERATION or SA_NTF_UNKNOWN_OPERATION	
Changed State Attribute ID	Optional	SA_MSG_DEST_CAPACITY_STATUS	30
Old Attribute Value	Optional	SA_MSG_QUEUE_GROUP_CAPACITY_REACHED	
New Attribute Value	Mandatory	SA_MSG_QUEUE_GROUP_CAPACITY_AVAILABLE	35


7	Message Service Management Interface	1
	Currently, an SNMP MIB interface is defined for the Message Service. Other man- agement access methods to the Message Service may be added in future versions of this specification.	5
7.1	Message Service MIB (SAF-MSG-SVC-MIB)	
	The Message Service MIB contains the three read-only tables <i>saMsgQueueTable</i> , <i>saMsgQueuePriorityTable</i> and <i>saMsgQueueGroupTable</i> , 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	10
	cluster.	15
	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.	20
	For a detailed description of the various objects of this MIB, refer to the SAF-MSG-SVC-MIB as can be downloaded from	
	<u>Intp://www.salorum.org/specification/download/get_spec</u> .	25



1

5
10
15
20
25
30
35

SERVICE **AILABILITY** FORUM

Index of Definitions

Α

acknowledgment see transmission acknowledgment at-most-once delivery property 20

в

broadcast distribution policy 18

С

capacity see critical capacity close time of a message queue 28 critical capacity 31

D

data see message data data size 30 data size see message data size delivery guarantees 20 distribution policies see also message queue groups definition 18 broadcast 18 local best queue 18 local equal load 18 multicast 18 unicast 17

н

high-water capacity thresholds 31

implementation-specific portion see message metadata integrity of messages 20

L

local best queue distribution policy 18 local equal load distribution policy 18 local member message queue 18 low-water capacity thresholds 31

Μ

message data 17, 31 message data size 17 message metadata see also messages definition 17 implementation-specific portion 17 standard portion 17 message priority 17, 20, 31 message queue groups see also distribution policies; messages; message queues definition 17 local member message queue 18 remote member message queue 18 message queues see also message queue groups; messages; priority areas definition 17 acknowledgment 21 at-most-once delivery property 20

close time 28 delivery guarantees 20 integrity of messages 20	1
nonpersistent 19 persistence of messages 21 persistent 19 retention time 27 message retention time 19 message size 17 message type 17, 30	5
message version 17, 30 messages <i>see also</i> message queues; message queue groups; mes- sage metadata definition 17 data 17, 31 data size 17, 30 message size 17	10
priority 17, 20, 31 reply 88 sender name 17, 31 type 17, 30 version 17, 30 metadata see message metadata multicast message queue group 18	15
N nonpersistent message queues 19	20
P persistence of messages 21 persistent message queues 19 priority areas <i>see also</i> message queues definition 17 capacity 31 critical capacity 31 high-water capacity thresholds 31 low-water capacity thresholds 31 priority <i>see</i> message priority	25
Q queue groups see message queue groups queues see message queues	30
R remote member message queue 18 reply buffer 24 reply message 88 retention time of a message queue 27	35
S sender id 23, 78, 89 sender name 17, 31 size <i>see</i> message data size; message size standard portion <i>see</i> message metadata	40
T thresholds see priority areas 31 transmission acknowledgment 21 type see message type	



U unicast message queue group 17	1
V version see message version	_
	5
	10
	15
	20
	25
	30
	35